State-of-the-Art Survey

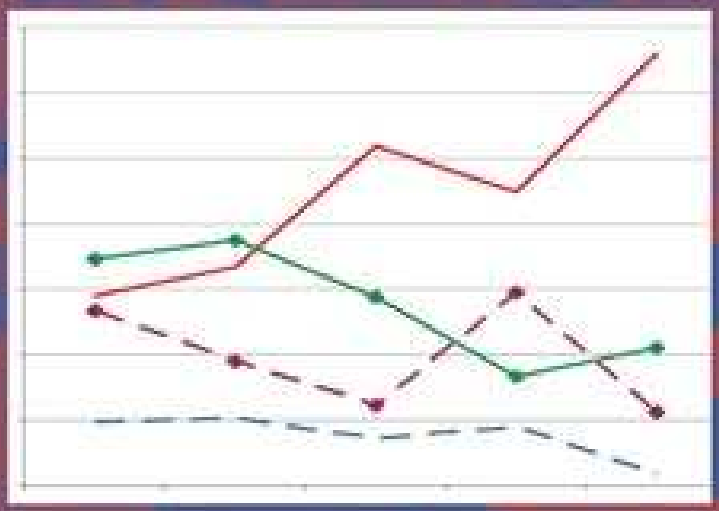
LNCS 4336

Victor R. Basili Dieter Rombach
Kurt Schneider Barbara Kitchenham
Dietmar Pfahl Richard W. Selby (Eds.)

Empirical Software Engineering Issues

Critical Assessment and Future Directions

International Workshop
Dagstuhl Castle, Germany, June 2006
Revised Papers



Springer

Lecture Notes in Computer Science 4336

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Victor R. Basili Dieter Rombach
Kurt Schneider Barbara Kitchenham
Dietmar Pfahl Richard W. Selby (Eds.)

Empirical Software Engineering Issues

Critical Assessment and Future Directions

International Workshop
Dagstuhl Castle, Germany, June 26-30, 2006
Revised Papers

Volume Editors

Victor R. Basili
Univ. of Maryland at College Park, 20742 College Park, MD, USA
E-mail: basili@cs.umd.edu

Dieter Rombach
Fraunhofer Institut, 67663 Kaiserslautern, Germany
E-mail: dieter.rombach@iese.fraunhofer.de

Kurt Schneider
University Hannover, 30167 Hannover, Germany
E-mail: kurt.schneider@inf.uni-hannover.de

Barbara Kitchenham
Keele University, ST5 5BG, Staffordshire, UK
E-mail: barbara.kitchenham@cs.keele.ac.uk

Dietmar Pfahl
University of Calgary, T2N 1N4 Calgary, Canada
E-mail: dpfahl@ucalgary.ca

Richard W. Selby
Northrop Grumman Space Technology, 90278 Redondo Beach, CA, USA
E-mail: rick.selby@ngc.com

Library of Congress Control Number: 2007922331

CR Subject Classification (1998): D.2, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-71300-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-71300-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12032105 06/3142 5 4 3 2 1 0

Preface

Victor R. Basili, Dieter Rombach, and Kurt Schneider

Introduction

In 1992, a Dagstuhl seminar was held on "Experimental Software Engineering Issues" (seminar no. 9238). Its goal was to discuss the state of the art of empirical software engineering (ESE) by assessing past accomplishments, raising open questions, and proposing a future research agenda.

Since 1992, the topic of ESE has been adopted more widely by academia as an interesting and promising research topic, and in industrial practice as a necessary infrastructure technology for goal-oriented, sustained process improvement. At the same time, the spectrum of methods applied in ESE has broadened. For example, in 1992, the empirical methods applied in software engineering were basically restricted to quantitative studies (mostly controlled experiments), whereas since then, a range of qualitative methods have been introduced, from observational to ethnographical studies. Thus, the field can be said to have moved from experimental to empirical software engineering.

We believe that it is now time to again bring together practitioners and researchers to identify both the progress made since 1992 and the most important challenges for the next five to ten years.

Objectives

The purpose of this workshop was to gather those members of the software engineering community who support an engineering approach, based on empirical studies, to:

- Identify the progress of ESE since 1992 (Dagstuhl Seminar No. 9238*)
- Summarize the state of the art in ESE
- Summarize the state of the practice in ESE in industry
- Develop an ESE roadmap for research, practice, education and training

Three sessions were set up to discuss some of the most eminent challenges in ESE:

1 The Empirical Paradigm (Lead: Dieter Rombach)

The two topics addressed in this session were "Approaches for Empirical Validation" and "Exploration versus Confirmation." The session was complemented by a historical review.

Approaches for Empirical Validation
Numerous types of approaches are used for empirical validation. Questions relevant in this context include:

Which approaches are useful in what situation? How do we combine quantitative and qualitative studies? Which new and innovative approaches have surfaced lately?

Exploration versus Confirmation
Empirical studies range from exploring new, badly understood software engineering approaches for the purpose of incremental learning to confirming well-defined software engineering approaches. This raises a number of questions:
Which of these study forms is appropriate under what circumstances? How do we deal with validity threads (especially for exploratory studies)?

Historical Review
Guest Speaker: Mike Mahoney, a science historian from Princeton, provided a historical review on how other disciplines have dealt with the issue of "exploratory versus confirmatory studies." He then suggested some ideas for software engineering.

2 Measurement and Model Building (Lead: Victor Basili)

The two topics addressed in this session were "Data Sharing" and "Effective Data Interpretation."

Data Sharing
We have not yet established clear rules for handling ownership of empirical data – like those in other disciplines such as physics. This raises the following questions:
What is the value of empirical data, testbeds, and other study artifacts? How should they be shared for replication? What are the limits of sharing? How should ownership be recognized in publications? How can the probability of misuse be minimized?

Effective Data Interpretation
Proper interpretation of empirical data is a challenging task. The most obvious questions in this context include:
What are scientifically acceptable means of interpretation? What new approaches (e.g., visualization, simulation) can help? How do we combine evidence from individual studies into more abstract evidence?

3 Technology Transfer and Education (Lead: Kurt Schneider)

The two topics addressed in this session were "Technology Transfer" and "Education." They are two facets of making an impact with empirical work.

Technology Transfer
Empirical studies can be used to facilitate and speed up technology transfer into practice. Relevant questions in this context include:
How do we package results for different purposes and contexts? How can we use empirical software engineering to speed up technology transfer? What information do practitioners need from empirical studies? What is most convincing for practitioners?

Education
Empirical methods have to be taught, on the one hand; on the other hand, empirical studies can be used to teach other computer science and software engineering topics better. This raises a number of questions:
What are the experiences for effectively teaching empirical study competence? What are the experiences and approaches for integrating empirical studies into computer science or software engineering curricula? What educational methods are suitable for teaching empirical or evidence-based software engineering? How can education be improved?

Workshop Organization

To address the aforementioned challenges, an international workshop on the topic of "Empirical Software Engineering" was organized and held at the International Conference and Research Center for Computer Science (IBFI) at Dagstuhl Castle in Germany. The motivation for this workshop was to provide a forum for a relatively small but representative group of leading experts in software engineering with an emphasis on empirical studies from both universities and industry to meet and reflect on past successes and failures, assess the current state of the practice and research, identify challenges, and define future directions. An Organizing Committee identified key topics and key people who should participate in the workshop. The topics were chosen for discussion along with the session chairs and the people who were to present introductory talks. A final session was aimed at devising a roadmap for future work.

After the selection of discussion topics, introductory talks, and session chairs, approximately 30 more participants were invited to submit position statements on the selected topics. The participants came from Europe, the USA and Canada, Asia, and Australia.

The workshop was scheduled to run from Tuesday, June 26, 2006, through noon on Friday, June 30, 2006. Three full-day sessions were devoted to the topics listed. A half-day session was devoted to the development of a roadmap.

Session Organization

During each session, the Chair set the stage, by summarizing the state of 1992 and introducing the two selected topics. Two introductory presentations per topic set the tone and raised issues for discussion by identifying the progress made since the 1992 Dagstuhl Workshop, raising issues and challenges for the next five to ten years, and providing a list of provocative statements. Based on the issues raised by the introductory presentations and additional issues raised by other seminar participants, lively discussions took place. The discussions were deepened in up to four parallel working groups. Each working group was asked to provide a summary of their discussion. The session was concluded with the presentations of the working groups and final discussions. The material contained in this volume includes, for each session, the keynote address, introductory talks, position papers, summaries of the working groups, and a discussion summary.

Results

In session 1, on the empirical paradigm, the topics of approaches for empirical validation and of exploration versus confirmation were discussed. Workshop participants agreed that the community has matured since 1992 but is still in a very early phase compared to other disciplines. Improvements were suggested, among others, with regard to types of studies (e.g., longitudinal case studies), complementary usage of quantitative and qualitative studies, and theories for aggregating results across studies.

In session 2, the topics of data sharing and effective data interpretation were discussed in the context of measurement and model building. Clearly, as a community we need to build on each other's work in order to build models that represent the knowledge of the discipline better. We need effective approaches to interpret these models across multiple domains, environments, contexts, etc. Workshop participants agreed on these goals but did not always agree on how to achieve them. Discussions ran from debates on the necessity of protocols, to the use of open source data as opportunities of study, to ways of combining the results from individual studies, to building theories based on multiple studies.

In session 3, technology transfer and education were discussed. These issues are tightly interwoven. Good empirical education will lead to competent graduates. In their industrial careers, they are more likely to use and adopt empirical results than others. However, this will be a long-term effect. Workshop participants agreed there need to be explicit, short-term technology transfer mechanisms. Discussions and working groups discussed achievements and pointed to future research agendas.

Past Achievements

Since 1992, the community's understanding of how to perform empirical studies in the area of SE has improved. Consequently, the amount of empirical study activity has grown dramatically and there are many more sources of data and results from studies than there were then. Furthermore, the community was able to provide a body of knowledge from empirical studies in a few software engineering areas (e.g., inspections).

There have been a fair number of collaborations, some sharing of data, attempts at combining evidence from various studies and even the beginnings of the development of theories. Some of these results come from the enlargement of the community of researchers and some come from the availability of vehicles for publication and sharing of knowledge, e.g., ISERN: the International Software Engineering Research Network (started in 1993), the *Journal of Empirical Software Engineering* (started in 1996), and the International Symposium on Empirical Software Engineering (started in 2001). Some of the data have begun to be shared by different research groups, often involving some form of collaboration. But there are problems associated with successful sharing, e.g., regarding the overhead. Open source has become a major source of new data and opportunity for study. Various mechanisms are being studied for combining results of individual studies using experience bases and evidence-based approaches. Very little has been done in building theories from multiple studies but the problem has been identified as an area for further research.

Since 1992, empirical methods have been taught at many universities today. All courses discussed were created and designed after the 1992 workshop. The necessity to study empirical work in software engineering was seen on different levels of education, from Bachelor to graduate levels. Today, there is a clear shift to expecting empirical validation of results in software engineering publications. Papers are less likely to be accepted without them. PhD students, the potential next generation of researchers, are exposed to ESE techniques in a growing number of institutions.

Empirical validations do not necessarily require controlled experiments. During the last few years, well-prepared case studies in a realistic environment added an important facet. In a real industry project, a technique can prove its scalability and its fit to a specific industry context. Therefore, collaborations between academia and industry have been widely acknowledged in the meantime. They are a key to making an impact with empirical results. This awareness has grown significantly since 1992.

Key Points of Dissent

Probably the biggest source of dissent was the need for protocols in the sharing of data. Some felt that there should be some organized set of protocols for sharing data that would promote the sharing activity and protect the integrity of the data, while others felt a more free form of exchange was more appropriate. The concept of whether the experimenter owned the data was hotly debated and the view that programs are often considered the intellectual property of the developer was considered a detriment to data sharing.

Much empirical work is based on identifying problems, goals, or hypotheses, and then identifying the kind of design that should be applied and data that should be collected. There was some concern that studying open source was more of a bottom-up approach and that the data needed might not always be available, leading to a poor study or a change of focus for the study to satisfy the data that are there.

Although there is a general trend towards offering courses on empirical techniques, there is no consent on their contents or structure. Depending on the teachers' attitude, courses either tend to be focused on statistical methods or on conveying a more general view of research and evidence. Different books on empirical results emphasize different styles. Stimulating reflection in students is the common driver of both attitudes.

Collaboration with industry is seen as an important prerequisite. Researchers want to transfer their results to industrial use. Often, this is associated with funding through a company. Many see consulting and industry validation almost as synonyms. However, some participants stressed the two-way character of such a collaboration. Not only will practitioners learn from researchers, but researchers will also need to learn from practice: What are relevant research questions? What are valid arguments to sell a result? Researchers and industry experts need to meet on middle ground.

Important Topics for Future Work

A historian's perspective yielded the insight that ESE should not feel that it is behind, but there is a normal progress in the maturing of science that, as history shows, takes some time; ESE still needs to probe / explore what the important factors are in

software engineering. Much of science rests on engineering experience and ESE needs to accept that wrong models can be useful to advance science.

Generalization remains a major issue for controlled experiments performed under laboratory conditions. Generalization can be approached by building (logical) models. Extend the body of knowledge from empirical studies by incorporating the whole range of empirical evidence, ranging from controlled experiments to longitudinal case studies, by also integrating quantitative and qualitative research methods.

To better support reuse of empirical knowledge (combination of results), standardized ways, not only for performing empirical studies but also for reporting the results, have to be agreed upon.

Combining the results of various studies is necessary to evolve our understanding of the software engineering discipline. There is a need to share data and artifacts across research groups so that results of varying studies can be analyzed, models built and evolved, and influencing variables identified. An article offering various protocols for the sharing of data and artifacts has been published in the *Journal of Empirical Software Engineering* (2007). The future work involved here is to test out the protocols and evolve them based on experience.

In many ways, empirical study in software engineering is opportunistic, and we need to find those opportunities for data and artifacts. Open source is one such opportunity. We need to experiment with mechanisms that allow us to use open source projects in more effective ways, e.g., by letting an empiricist be part of the study from the beginning.

Identifying and experimenting with ways to combine data from multiple studies are one of the most important areas of research if we are to be successful in building effective models in the software engineering discipline. We still struggle with how to do it effectively.

Abstracting from models to theories is a natural progression for most disciplines. Research is needed in approaches to building theories, representing them, documenting them, schematizing them, and evaluating them.

Teaching empirical techniques requires good teaching material, such as textbooks. Sample documents and free data would be of great help to try established empirical techniques. Participants suggested exchanging study material on specific Web sites. These could be shared among the empirical research community.

Establishing and maintaining collaborations between industry and academia will be even more important in the future. Many (junior) researchers find it challenging to approach a company successfully. Even senior researchers admit there can be many failures and disappointments before a fruitful collaboration emerges. More guidance and more examples in the area of collaborations would lower the threshold.

How to select rewarding topics for ESE research was discussed several times. While research had important successes in fields such as reviews and inspections in the past, many believe the agenda needs to be broadened. Different schemes were proposed to visualize current areas of work. Industry will have a say in defining future research roadmaps.

Established, well-understood areas of empirical work, such as inspections, make good candidates for a common repository of education material. Every student who wants to become a mature empirical researcher should replicate some of the classical inspection studies!

Acknowledgements

It would have been difficult to organize a workshop like this without help from the community. First, we would like to thank all the workshop participants for their effort in submitting position papers and participating in the discussions. Next, the presenters of the introductory talks and the session chairs as well as the chairs of the working groups deserve a special thank you for their contributions to the success of the workshop.

Clearly, an effort such as this could not have been successful without financial support. We would like to acknowledge and sincerely thank IBFI for supporting accommodation and providing excellent meeting facilities and the Fraunhofer Institute for Experimental Software Engineering (IESE) for additional general purpose funds. Our special thanks go to Marcus Ciolkowski from University Kaiserslautern, Sira Vegas from the Universidad Politécnica de Madrid, and Andreas Jedlitschka from Fraunhofer IESE for having taken notes of all discussion sessions. Furthermore, we would like to thank Andreas Jedlitschka and Marcus Ciolkowski for their splendid job in providing organizational support, arranging the social event, and last but not least for editing these proceedings.

September 2006

Victor R. Basili
Dieter Rombach
Kurt Schneider

Reference

[1] Rombach, H.D., Basili, V.R., Selby, R.W. (Eds.): Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop Dagstuhl Castle, Germany, September 1992, Springer Verlag, LNCS 706

Table of Contents

The Empirical Paradigm
Introduction

Dieter Rombach

Abstract. The session on the empirical paradigm focused on two main aspects, (1) approaches for empirical validation and (2) exploration versus confirmation. The main conclusions are that the community has matured since 1992 but is still in a very early phase compared to other disciplines. Improvements were suggested, among others, with regard to realism of studies, quality of studies, and complementary usage of quantitative and qualitative methods.

1 Introduction

The session on the empirical paradigm emphasized the following topics: (1) Approaches for empirical validation, and (2) Exploration version confirmation.

2 Approaches for Empirical Validation

"Approaches for Empirical Validation" aimed at discussing the numerous types of approaches that are used for empirical validation. The following questions were used to initiate a discussion.

- Which approaches are useful in what situation?
- How do we combine quantitative and qualitative studies?
- Which new and innovative approaches have surfaced lately?

The two introductory talks of this session were given by Marvin Zelkowitz on "Techniques for Empirical Validation" and by Walter Tichy on "Status of Empirical Research in Software Engineering".

The topic "Approaches for Empirical Validation" is complemented by short papers by Marcus Ciolkowski on the aggregation of empirical evidence, by Lionel Briand on empirical evaluation in software engineering: role, strategy, and limitations, by Markku Oivo on new opportunities for empirical research, by Carolyn Seaman on the empirical paradigm, and by Austen Rainer on the value of empirical evidence for practitioners and researchers.

3 Exploration Versus Confirmation

"Exploration versus Confirmation" aimed at discussing the use of empirical studies as they range from exploring new, badly understood software engineering approaches

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 1 – 3, 2007.

for the purpose of incremental learning to confirming well-defined software engineering approaches. The following questions were used to initiate a discussion:

- Which of these study forms is appropriate under what circumstances?
- How do we deal with validity threads (especially for exploratory studies)?

The two introductory talks were given by Victor Basili on "The Role of Controlled Experiments in Software Engineering Research" and by Barbara Kitchenham on "Empirical Software Engineering – Problems and Opportunities".

The topic "Exploration versus Confirmation" is complemented by short papers by James Miller on creating real value in software engineering experiments, by Guilherme Horta Travassos, who discussed issues from silver bullets to philosophers' stones, by Helen Sharp on social and human aspects of software engineering, by Tracy Hall on longitudinal studies in evidence-based software engineering, and by Jeffrey Carver on the use of grounded theory in empirical software engineering.

4 Working Groups

Four working groups were formed to address issues raised in previous discussions:

- "Increase reputation of empirical research & domains of study", led by Larry Votta.
- "Return on Investment for empirical research - most valuable insights so far", led by Lutz Prechelt.
- "Combination of different types of studies for more realistic investigations" (such as controlled experiments and case studies), led by C. Seaman.
- "Role of controlled experiments", led by Lionel Briand and Andreas Jedlitschka.

5 A Historical Perspective

Mike Mahoney, a science historian from Princeton, provided a historical review on how other disciplines have dealt with the issue of "exploratory versus confirmatory studies". He suggested some ideas for software engineering.

6 Results

The main results of this session are:

- There is still room for improving the marketing of empirical software engineering, e.g., by providing an empirically based software engineering handbook.
- In order to complement findings from quantitative analysis, more emphasis should be placed on qualitative methods.
- The community should strive for
  - more realistic studies, e.g., industrial project observations, and
  - a common research agenda.

- The quality of studies, from design to reporting (incl. publication), shall be improved further.

As presented by Mike Mahoney in his historical perspective, where he compared empirical software engineering with other disciplines, empirical SE should not feel that it is behind, but that there is normal progress in the maturing of science; empirical software engineering is still in a phase where it needs to probe and explore what the important factors are.

Techniques for Empirical Validation

Marvin V. Zelkowitz

Abstract. In 1998 a survey was published on the extent to which software engineering papers validate the claims made in those papers. The survey looked at publications in 1985, 1990 and 1995. This current paper updates that survey with data from 2000 and 2005. The basic conclusion is that the situation is improving. One earlier complaint that access to data repositories was difficult is becoming less prevalent and the percentage of papers including validation is increasing.

1 Introduction

Any science advances by the process of developing new abstract models and then a series of experiments to test those models against reality. However, all too often in the software engineering domain, models (e.g., programs, theories) are described without any corresponding validation that those models have any basis in reality.

In order to determine the status of experimental validation in software engineering, in 1998 a paper by Zelkowitz and Wallace [4] surveyed the research literature in order to classify the experimental methods used by authors to validate any technical claims made in those papers. A total of 612 papers, published in 1985, 1990 and 1995, were studied. Of these, 62 were deemed not applicable, leaving 560 research papers. The 3 data sources used for this survey were:

- ICSE – Proceedings of the International Conference on Software Engineering
- TSE – IEEE Transactions on Software Engineering
- SW – IEEE Software Magazine

Each of the research papers was classified according to a 14-scale taxonomy:

1. Project monitoring. Collect the usual accounting data from a project and then study it.
2. Case study. Collect detailed project data to determine if the developed product is easier to produce than similar projects in the past.
3. Field study. Monitor several projects to collect data on impact of the technology (e.g., survey).
4. Literature search. Evaluate published studies that analyze the behavior of similar tools.
5. Legacy data. Evaluate data from a previously-completed project to see if technology was effective.
6. Lessons learned. Perform a qualitative analysis on a completed project to see if technology had an impact on the project.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 4 – 9, 2007.

7. Static analysis. Use a control flow analysis tool on the completed project or tool.
8. Replicated experiment. Develop multiple instances of a project in order to measure differences.
9. Synthetic. Replicate a simpler version of the technology in a laboratory to see its effect.
10. Dynamic analysis. Execute a program using actual data to compare performance with other solutions to the problem.
11. Simulation. Generate data randomly according to a theoretical distribution to determine effectiveness of the technology.
12. Theoretical. Formal axiom-proof style paper describing a new theory.
13. Assertion. Informal feasibility study of the technology. (More of an existence proof rather than an evaluation of the claims of the technology).
14. No experimentation. The default classification if a paper fails to fall into any of the preceding classification.

The first eleven categories represented various empirical validation methods. Method 12 (Theoretical) indicated that the paper was a formal model of some property. (The original 1998 paper did not include a separate theoretical category, as methods 12 and 14 were combined as one category.) There was a thirteenth quasi-validation method, called an assertion. Assertion papers were those where the author knew that an experimental validation would be appropriate, but only a weak form of validation was applied. (For example, a paper describing a new programming language might only show that it was feasible to write programs in that language, not whether the programming language solved any underlying problem that needed to be solved.) All other papers were characterized as "No experimentation," indicating that some form of validation was appropriate, but was lacking.

The basic conclusion was that approximately half of the papers had an inadequate level of validation. Similarly, Walter Tichy in 1994 [2] did his own literature search using a different protocol, yet came up with a similar conclusion. The general result was that the software engineering community was not doing a good job in developing a science of software development.

It is now ten years after these two surveys, so it seemed appropriate to redo the 1998 study in order to see if the situation had changed. One of the conclusions in the Zelkowitz and Wallace paper was that the situation seemed to be improving. Since two more 5-year milestones have since passed, it is worthwhile to revisit that initial survey to see how the research world has changed in the approximately 10 years since the original survey was conducted.

If we were to redo in total, a slightly different taxonomy would be chosen than the 14-point scale given above. However, one goal was to understand how the research world has changed since the 1990s, so the same classification model was used. One problem today is that we don't have an agreed upon model for classifying software engineering research methods. Two other surveys compiled in the interim period [1] [3] use a different classification model for determining the experimental validation method used.

Table 1 presents the basic data from both the original and 2006 survey. In the 2006 survey, an additional 361 papers were evaluated, with 35 not applicable, leaving 326 additional research papers to classify.

2 Observations

The percentages (excluding the "not applicable" category) for each of the 13 validation methods are given in Figure 1. Case study remains the most popular method, increasing in each survey period from 8.3% of the papers in 1985 to 18.8% in 2005. The "classical" experimentation method of a controlled replicated study (represented as the sum of synthetic and replicated in Figure 1) grew slightly to 5.3% of the papers in 2005 from 2.6% in 1985. Dynamic analysis dominated the experimental methods in 2005 with 20% of the papers. A possible reason why this is so is given later.

Table 1. Classification data from 973 papers: 1985-2005

	Project monitoring	Case study	Field study	Literature search	Legacy	Lessons learned	Static analysis	Replicated	Synthetic	Dynamic analysis	Simulation	Assertion	Theoretical	No experimentation	Not applicable	Total
ICSE	0	5	1	1	1	7	1	1	3	0	2	12	3	13	6	56
TSE	0	12	1	3	2	4	1	0	1	0	10	54	18	38	3	147
SW	0	2	0	1	1	5	0	0	1	0	0	13	1	10	6	40
1985 Total	0	19	2	5	4	16	2	1	5	0	12	79	22	61	15	243
ICSE	0	7	0	1	2	1	0	0	0	0	0	12	1	7	4	35
TSE	0	6	1	1	2	8	0	1	4	3	11	42	19	22	2	122
SW	1	6	0	5	0	4	0	0	1	0	0	19	0	8	16	60
1990 Total	1	19	1	7	4	13	0	1	5	3	11	73	20	37	22	217
ICSE	0	4	1	0	1	5	0	1	0	0	1	4	3	7	5	32
TSE	0	10	2	2	1	8	2	3	2	4	6	22	7	7	1	77
SW	0	6	1	3	1	7	0	0	0	0	1	14	0	3	7	43
1995 Total	0	20	4	5	3	20	2	4	2	4	8	40	10	17	13	152
ICSE	0	10	0	0	1	4	0	2	2	4	1	11	3	20	10	68
TSE	0	9	3	1	0	0	0	0	4	4	7	11	10	15	2	66
SW	0	7	3	1	1	3	0	0	3	0	0	4	1	11	19	53
2000 Total	0	26	6	2	2	7	0	2	9	8	8	26	14	46	31	187
ICSE	0	14	1	0	1	0	0	0	3	8	1	10	1	3	0	42
TSE	0	9	4	1	5	0	2	1	2	13	5	13	1	8	2	66
SW	0	9	4	1	5	0	2	1	2	13	5	13	1	8	2	66
2005 Total	0	32	9	2	11	0	4	2	7	34	11	36	3	19	4	174

More important than individual methods is the general "health" of the software engineering research field. This is summarized by Figure 2. Except for 2000, the percent of "No experimentation" papers dropped from 26.8% in 1985 to only 10.9% in 2005. Assertions dropped from 34.6% in 1985 to 21.2% in 2005. The percent of papers that used one of the 11 validation methods rose from 29% to 66% in 2005. (The percentage rose from 39% to 68% when theoretical papers were also included.) Clearly the situation is improving. This is consistent with an alternative study of the International Software Engineering conferences (ICSE) [3]. Using a sampling technique over all 29 ICSE proceedings, they found that 19 of 63 papers included no empirical study (30%). This present study indicates that 50 out of 208 ICSE papers

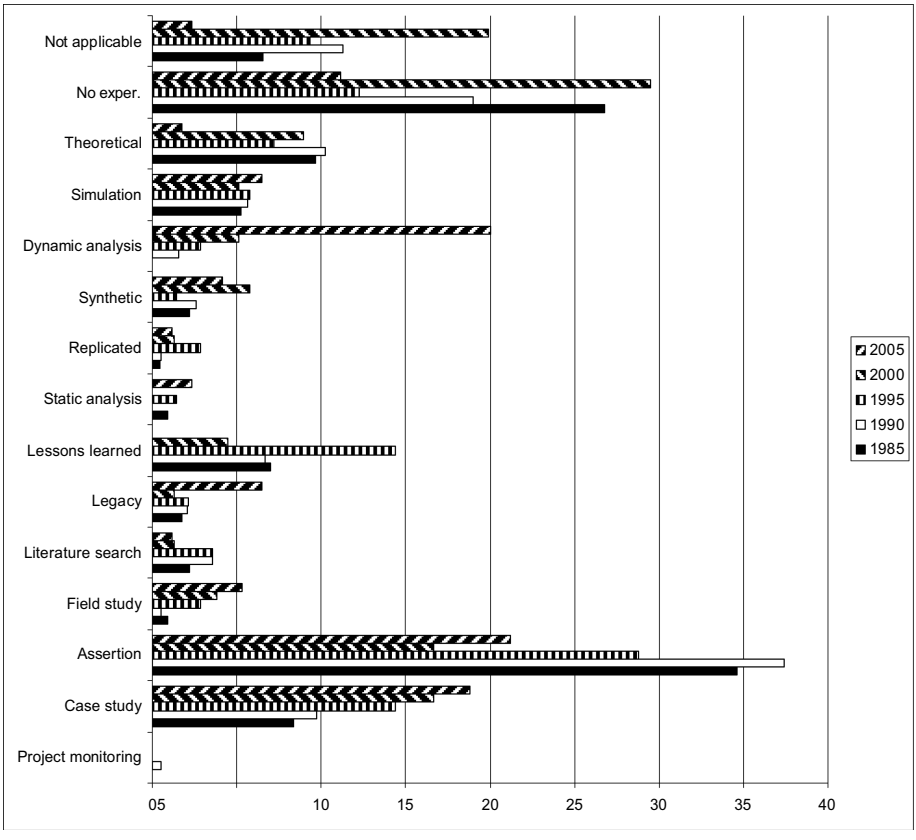


Fig. 1. Percentages of each validation method

(26%) had no experimentation. They also found a statistically significant increase in evaluation papers between the conferences prior to 1990 and those since then.

Unlike in the Zannier et al study [3], no attempt was made to evaluate the quality of the validation presented in those papers. (It was beyond our knowledge to understand and evaluate all 886 papers, but it was fairly easy to understand hoe the authors proposed to evaluate that technology.) If the paper stated an hypothesis about the technology described in the paper (even if stated indirectly) and then proceeded to describe a validation method for that hypothesis, we considered it as validated. Perhaps the hardest part of the study was trying to understand what the underlying hypothesis really was and how the authors would proceed to evaluate it. As stated earlier, we need a common terminology in which to describe validation methods. Many of the authors used terms like “experiment,” “case study,” “simulation,” “controlled,” etc. in very different ways.

Several anecdotal observations are buried in the data. A common complaint 20 years ago was the lack of published data sources that others could have access to. That seems to be changing. Many of the papers used the various open source

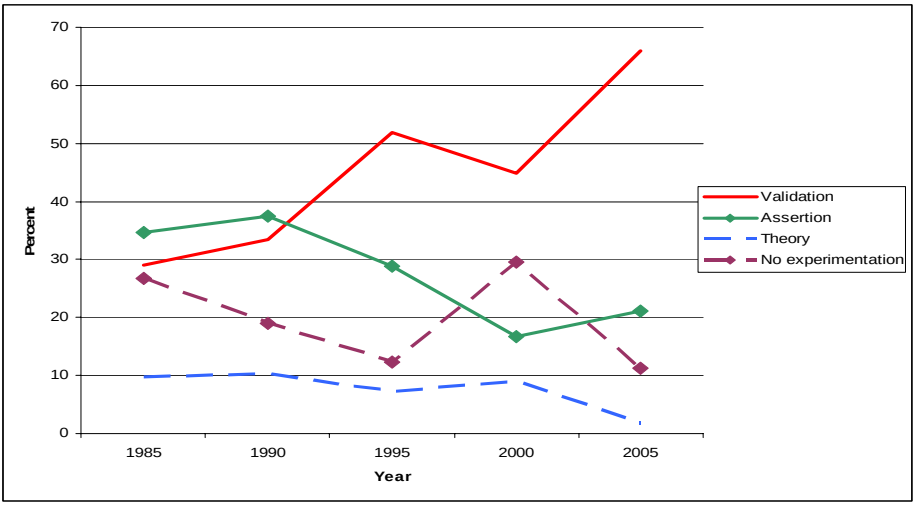


Fig. 2. Changes over time in validated papers

repositories, looking at the development history of products such as the Apache web server or Mozilla, as sources for data. This use of historical data using open source and other data repositories was one of the reasons for the rise in the dynamic analysis category in Figure 1. Similarly, data mining through theses sources led to the rise in the legacy data category.

3 Conclusions

There are several threats to the validity of this study.

1. The 2006 classification was performed about 9 years after the earlier study. While the same classification process was used to classify the papers according to the 14-point taxonomy, undoubtedly the intervening years may have changed our views of some of the validation methods. Consistency of this somewhat subjective classification method is a problem. For example, in [1], they report 0 and 3 controlled studies in ICSE for 1995 and 2000, respectively, while Table 1 shows 1 and 4, respectively, for those years in our classification). While this may have affected individual percentages in Figure 1, it should not have had much of an impact on the overall results as given by Figure 2.

2. As with the earlier 1998 study, each paper source for each year was managed by a different editor or conference chair. This has an effect on the overall acceptance rate of various papers submitted to that source. For example, the rise in "No experimentation" in 2000 was partially due to the largest number of ICSE papers (68) in the entire survey and the relatively large number of "No experimentation" papers (20) in those proceedings. Although such variances affect individual sources in a given year, the overall trends seem consistent.

3. There was a change in the scope of IEEE Software between 1995 and 2000. In the earlier survey, this magazine often published longer articles that had a research component. However, more recently the papers have been shorter with more regular columns appearing in each issue. Regular columns were not included in this survey and a value judgment was made on the remainder of the papers. If a paper discussed many solutions to a given problem, the paper was considered a tutorial or survey and listed as "Not applicable," but if the paper focused on a particular technique (often the author's), then it was considered a research paper.

The greatest limitation to this study, however, was mentioned earlier – the quality of the evaluation was not considered in classifying a paper. If the field is to mature as a scientific discipline, not only do we need empirical validation of new technology, we also need quality evaluations. However, that study still needs to be done.

In spite of these limitations, the results should prove of interest to the community. It provides a general overview of the forms of validation generally used by the computer science community to validate the various research results that are published and it does show that the field is maturing. Computer science seems to be developing an empirical culture so necessary to allow it to mature as a scientific discipline.

References

1. Sjøberg D. I. K., J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N-K Liborg and A. C. Rekdal, A survey of controlled experiments in software engineering, IEEE Trans. on Soft. Eng. 31, 9 (2005) 733-753.
2. Tichy W. F., P. Lukowicz, L. Prechelt, and E. A. Heinz, Experimental evaluation in computer science: A quantitative study, J. of Systems and Software 28, 1 (1995) 9-18.
3. Zannier C., G. Melnik and F. Maurer, On the success of empirical studies in the International Conference on Software Engineering, Inter. Conf. on Software Eng., Shanghai, China (2006) 341-350.
4. Zelkowitz M. V. and D. Wallace, Experimental models for validating computer technology, IEEE Computer 31, 5 (May, 1998) 23-31.

Status of Empirical Research in Software Engineering

Andreas Höfer and Walter F. Tichy

Abstract. We provide an assessment of the status of empirical software research by analyzing all refereed articles that appeared in the Journal of Empirical Software Engineering from its first issue in January 1996 through June 2006. The journal publishes empirical software research exclusively and it is the only journal to do so. The main findings are: 1. The dominant empirical methods are experiments and case studies. Other methods (correlational studies, meta analysis, surveys, descriptive approaches, ex post facto studies) occur infrequently; long-term studies are missing. About a quarter of the experiments are replications. 2. Professionals are used somewhat more frequently than students as subjects. 3. The dominant topics studied are measurement/metrics and tools/methods/frameworks. Metrics research is dominated by correlational and case studies without any experiments. 4. Important topics are underrepr-esented or absent, for example: programming languages, model driven developpment, formal methods, and others. The narrow focus on a few empirically researched topics is in contrast to the broad scope of software research.

1 Introduction

During the 10½ years that have elapsed since the first issue of Empirical Software Engineering (ESE) appeared in January 1996, the journal has become the major venue for publishing empirical results in software research. It is the only journal exclusively dedicated to empirical studies in software. Thus, ESE can be seen as a good indicator for the status and health of empirical software research. We wanted to know what topics are addressed by empirical research, which research methods are used, and where the data comes from. Further, we were interested in the question whether there are important topics that are insufficiently covered by empirical research. To answer these questions, we performed an in-depth bibliographic study of all reviewed articles in ESE from volume 1, number 1 to volume 11, number 2.

2 Related Work

In 2005 Segal et al. [4] presented a study that investigated the nature of the empirical evidence reported in 119 papers which appeared in ESE between 1997 and 2003. The classification scheme used in this paper is based on the one developed by Glass et al. [2]. Segal et al. [4] found among other things, that about half of the papers focused on measurement/metrics and inspections/reviews, that authors were almost as interested in formulating as in evaluating, and that other disciplines are referenced rarely.

The classification scheme introduced by Glass et al. [2] differentiates papers in the field of computing based on five characteristics: topic, research approach, research

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 10 – 19, 2007.

method, reference discipline, and level of analysis. Glass and his colleagues applied the scheme to 369 articles published in six leading software engineering journals over the period from 1995 to 1999. They conclude that software engineering research is diverse in topic but narrow in its research approach and method. Glass also found that 98 % of the papers examined do not reference another discipline.

Zelkowitz and Wallace [6] define a taxonomy for the classification of papers within the field of software engineering. They classified 612 articles published during the years 1985, 1990, and 1995 in the journals IEEE Transactions on Software Engineering and IEEE Software as well as in the proceedings of the International Conference on Software Engineering. One of their findings is that about 30 % of all classified papers lack experimental validation, but note that this situation is improving.

Sjøberg et al. [5] selected controlled experiments from 5,434 articles published in nine journals (including ESE) and proceedings of three conferences. The 103 papers describing controlled experiments were characterized according to topic, subjects, tasks, and environment of the experiment. One of the main results of Sjøberg and his co-authors is that controlled experiments constitute only a small fraction (1.9 %) of articles published.

Lukowicz et al. [3] compare the percentage of papers with experimental validation in several computer science journals and conference proceedings to the percentage of experimental work in the two journals Neural Computation and Optical Engineering. The findings of this study, which classified 403 articles, indicate that there is a lack of empirical validation in computer science.

The present paper concentrates on empirical work in software engineering in the journal dedicated to this type of work and attempts to get an indication of research quality and breadth. It is closest to the work of Segal et al. [4], but surveys a longer time span, classifies research method according to accepted categories in psychological research, and divides the largest of the categories in the work by Segal et al. [4], software life-cycle, into subcategories. We also identify gaps in the coverage of research topics.

3 Research Method

3.1 Selection of the Articles

We gathered all issues of ESE from January 1996 to June 2006 and selected all reviewed articles. Titles, authors, and keywords of those papers were entered into a table for classification. We deliberately excluded 50 editorials, 30 viewpoints/position papers, 15 conference and workshop reports, and 6 comments/correspondence papers from the literature analysis. In total, we selected 133 reviewed articles.

3.2 Classification of the Articles

In order to develop a classification scheme for the articles, the authors jointly studied titles, keywords, and abstracts of all the articles that appeared in the first year of ESE. Out of this study, a first version of the classification scheme was developed. This scheme was refined during the classification process. Each paper was classified according to the three dimensions topic, method, and source of data.

Table 1. Topics

Topic
Design/Architecture
Diagrams/Notations
Empirical methods
Inspections/Reviews
Maintenance
Measurement/Metrics
Project planning/Estimation
Quality estimation/Fault prediction
Requirements
Software engineering process
Testing
Tools/Methods/Frameworks
Other

- **Topic:** The subject area of the paper within software engineering. Table 1 provides the list of topics. The categories are self-explanatory, except for the following:
    - The category Empirical methods covers tools or approaches to conduct empirical work; such papers aim to improve research methods.
    - There are categories for all major phases in software development (Design/Architecture, Inspections/Reviews, Maintenance, etc.), except implementation (this class is empty). The class Software engineering process includes papers that address more than one phase (usually the overall software development process).
    - The class Tools/Methods/Frameworks covers papers that introduce a novel tool, method or framework for software development, coupled with an empirical study (typically a case study).
- **Method:** The empirical research method used for the study. We use categories from psychological research according to Christensen [1]. We only present non-empty categories[1]: case study, correlational study, ethnography, experiment, ex post facto study, meta analysis, phenomenology, survey (see Table 2). Papers were classified according to the main method. For example, if a paper contains a survey as a preliminary step for an experiment, then it would be classified as experiment.
- **Source of data:** This characteristic categorizes the origin of the data used for empirical research (see Table 3).

The following topics were sub-classified: Empirical method, Measurement/Metrics, and Tools/Methods/Frameworks. The reason is that papers in these classes typically address an additional topic. For instance, an empirical method might be specific to

[1] Empty categories are: Longitudinal and cross-sectional study, naturalistic observation.

project planning, a metrics paper might apply to fault prediction, or a tool might be specific to the topic Requirements. Instead of double classification (which would be the alternative), we show the subcategories separately, in order to make the distribution of topics more transparent.

Table 2. Research Method

Method	Definition
Case study	In-depth analysis of a particular project, event, organization, etc.
Correlational study	Measuring variables and determining the degree of relationship that exists between them.
Ethnography	Description and interpretation of the culture of a group of people.
Ex post facto study	Study in which the variables of interest are not subject to direct manipulation, but must be chosen after the fact (e.g., when analyzing software repositories).
Experiment	Quantitative study to test cause-and-effect relationships.
Meta analysis	Integrates and/or describes the results of several studies.
Phenomenology	Description of an individual's or a group's experience of a phenomenon.
Survey	Data is collected by interviewing a representative sample of some population.

Table 3. Sources of Data

Source	Definition
Professionals	Data acquired from professionals directly by using them as subjects in an experiment or indirectly by collecting data from projects with professionals.
Students	Data acquired from students directly by using them as subjects in an experiment or indirectly by collecting data from a project with students.
Both	Data acquired from students and professionals.
Benchmarks	Benchmarks are artificially composed data designed to measure the performance of a tool, method, algorithm, etc.
Software	The source Software refers to data derived from operational software (such as reliability data) irrespective of the methods of development for such software.
Studies	Data acquired from other studies (meta analysis).
Unknown	Unstated source of data. Some articles do not state how the data was gathered or whether their subjects were students or professionals.

The classification process worked as follows. The first author initially classified all papers. If title, keywords, abstracts, and conclusions were not sufficient for classification, the whole article was studied. Doubtful assignments were tagged for the second author. After the first author had classified all articles, the second author checked the classification table for plausibility, spot-checked classifications in detail, and tagged additional doubtful classifications. The tagged classifications were then re-checked together and corrected if necessary.

4 Findings

4.1 Topic

Figure 1 depicts the distribution of topics. This dimension is dominated by the categories Measurement/Metrics and Tools/Methods/Frameworks followed by Inspections/Reviews and Software engineering process. The rest are all below 10 %. The categories Usability and Reliability were under 2 %, so we combined them with the papers that did not fit any category (class Other).

As mentioned, several categories have subtopics, which are not included in Figure 1. Of the 22 papers in the Measurement/Metrics category, half dealt with Project planning/Estimation and 27.3 % with Quality estimation/Fault prediction. Other topics are each under 5 %.

There are 20 Tools/Methods/Frameworks papers, but the topics are more spread out: 25.0 % Software engineering process, 20.0 % Quality estimation/Fault prediction, 15.0 % Project planning/Estimation, and 10.0 % Usability. The class Empirical methods contains 11 papers, with 36.4 % General (no particular topic) and 27.3 % dealing with project planning.

4.2 Research Method of the Papers Surveyed

The preferred research methods are Experiment and Case study (see Figure 2). Among the 50 papers describing an experiment 13 (26.0 %) were replications.

An interesting question is what methods were used in the top three topics. Among the 22 Measurement/Metrics papers, 36.4 % use correlational studies and 31.8 % case studies; there are no experiments and thus no systematic inquiries into cause and effect. For Tools/Methods/Frameworks, 55.0 % of 20 papers employ case studies, and 25.0 % experiments. Of the 17 articles with the topic Inspections/Reviews, 15 (88.2 %) use experiments, the remaining two papers contain case studies. Studies of Inspections/Reviews have the largest number of experiments. Diagrams/Notations is next with 7, followed by Design/Architecture with 6, and Project planning/Estimation as well as Tools/Methods/Frameworks each with 5. The high proportion of Inspections/Reviews combined with a high rate of experiments reflects the maturity of this area, as researchers are exploring causal relationships.

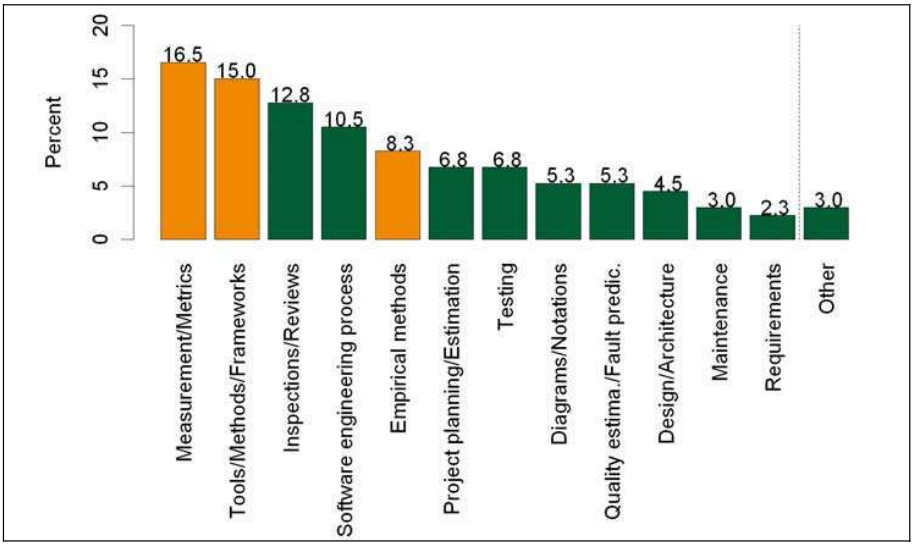


Fig. 1. Topic (Categories with subtopics are highlighted in orange.)

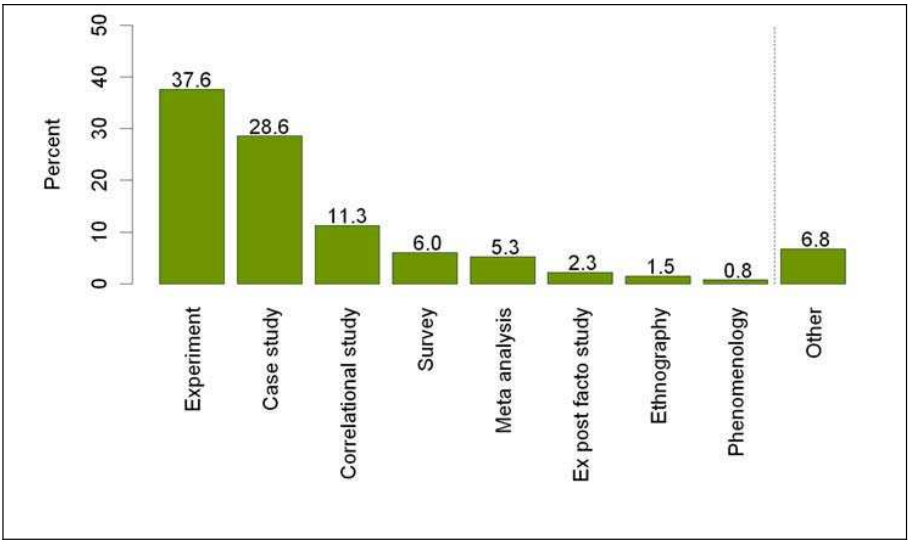


Fig. 2. Research Method

4.3 Source of Data

Figure 3 shows the source of data. Papers employing professionals and students dominate. In 63 publications, professionals only were used, and solely students in 36.

There were 10 papers using both, for example comparing professionals and students. Figure 4 shows a cumulative graph of the distribution of papers with professional and student subjects. Though the proportion of papers with students and professional subjects varies from year to year, it can be seen that cumulatively, articles using professionals outnumber those using students over the years.

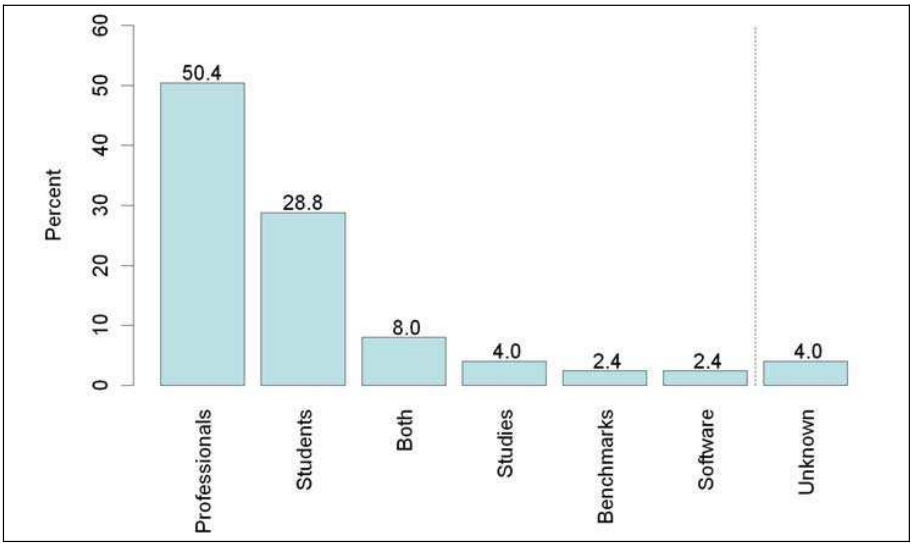


Fig. 3. Source of Data

As empirical work is often criticized for relying on students, we looked at the data source with respect to research method. It turns out that 78.9 % of case studies used professionals, 5.3 % students, and 2.6 % both. The situation is nearly reversed for experiments: 60.0 % used students, 22.0 % professionals, and 14.0 % used both students and professionals (see also Table 4). These findings are in line with those of Sjøberg et al. [5]. On a much larger sample of experiments, Sjøberg and his co-authors report that 72.6 % of experiments employed students, 18.6 % professionals and 8.0 % both.

Table 4. Proportion of Professionals and Students in the Top Three Research Methods

Type of Study	% (Number of Papers)		
	Professionals	**Students**	**Both**
Experiment	22.0 (11)	60.0 (30)	14.0 (7)
Case study	78.9 (30)	5.3 (2)	2.6 (1)
Correlational study	66.7 (10)	13.3 (2)	13.3 (2)
All types	50.4 (63)	28.8 (36)	8.0 (10)

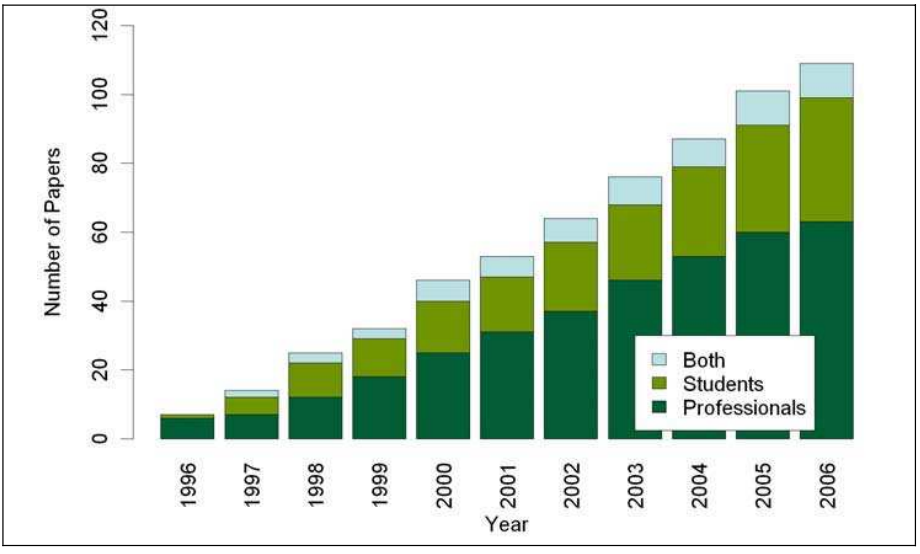


Fig. 4. Data from Students and Professionals

5 What is Missing?

Overall, it is a positive sign that studies with professionals outnumber those with students by a healthy margin. However, in experiments, student subjects dominate. This situation may reflect the difficulties of conducting controlled experiments outside a laboratory. More effort should be expended to repeat important experiments with professionals in order to improve generalizability of the results.

The Measurement/Metrics area is dominated by case studies and correlational studies, without any experiment. The lack of research into cause and effect seems to be a major weakness. It is well known that a correlation between two variables does not constitute a causal relationship; the values of both of these variables may be determined by other, hidden variables. There is strong evidence that causal relationships have not been identified: It is straight-forward for programmers to corrupt the indicator variables used today and thereby subvert any prediction based on them. By contrast, in the software inspections area, which is about as old as the metrics area, researchers have developed experimental techniques to successfully explore causal relationships.

Overall, the range of software topics studied empirically is rather narrow. Some important topics are missing completely. In particular, studies about programming languages and programming paradigms are conspicuously absent. As these topics are obviously important and subject to intense debate, studies comparing imperative vs. functional vs. scripting vs. object oriented languages are urgently needed, to inform further development of these languages and enable rational choices. Also missing are studies that compare programming approaches with standardized software that

substitute customization for programming. Program verification is not represented, but if verification is a practical approach, even in a limited domain, empirical studies are needed to determine efficacy. Absent were articles covering recent areas such as model driven development or aspect orientation. Furthermore, we expected to find papers illuminating the relationship between developer's personal characteristics and their optimal mode of work. Such studies would require collaboration with other disciplines such as social sciences and psychology, but references from software engineering to these disciplines are rare, as observed by Glass et al. [2] and Segal et al. [4]. Long-term studies of programming methods, such as agile methods were missing, too. Large gaps as to topic are confirmed by Sjøberg et al. [5] and Segal et al [4].

A discussion with participants of the Dagstuhl seminar brought up additional topics that are missing. Unclear are the feasibility bounds of techniques—i.e., determining in what situation or in what context a particular method or approach is preferable to another. Closely related are cost/benefit tradeoffs covering the development cycle, for example models for determining the relative effort to be spent on requirements, design, quality assurance, and so on. In other words, what is needed is an answer to the question of what has to be done when, and how much of it. A unifying theory about defect causation and detection would help guide quality assurance efforts. Finally, the grand challenge for software research was seen as developing an understanding of which software methods work and why. Such an understanding should provide a suitable foundation for predictable software processes and products.

6 Threats to Validity

The first threat to validity concerns the fact that articles reporting on empirical work are published in other venues as well. Thus, ESE might not provide a representative sample of all empirical research. But Sjøberg et al. [5] confirm some of our findings on a larger sample, restricted to controlled experiments.

We guarded against classification errors by a careful definition of classes and a cross check by a second person as described in section 3.2. Nevertheless, there are some borderline cases, and other raters might classify differently.

7 Conclusions

We conducted a literature review of all refereed articles published in ESE within the period form January 1996 to June 2006. We found that the use of professionals in 78.9 % of case studies is encouraging, while controlled experiments are predominantly conducted with students. The range of topics continues to be narrow and should be broadened considerably. The metrics area would benefit from emphasizing investigations into cause-and-effect relationships. The area of inspections and reviews appears to be methodologically mature with a high proportion of experiments.

References

1. L. B. Christensen, Experimental Methodology, 8th Edition, Allyn and Bacon (2001).
2. R. L. Glass, I. Vessey, V. Ramesh. Research in Software Engineering: An Analysis of the Literature. Journal of Information and Software Technology, vol. 44, no. 8, pp. 491-506 (2002).
3. P. Lukowicz, E. A. Heinz, L. Prechelt, W. F. Tichy. Experimental Evaluation in Computer Science: A Quantitative Study. Journal of Systems and Software, vol. 28, no. 1, pp. 9-18 (1995).
4. J. Segal, A. Grinyer, H. Sharp. The type of evidence produced by empirical software engineers. REBSE '05: Proceedings of the 2005 workshop on Realising evidence-based software engineering, pp. 1-4 (2005).
5. D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N-K Liborg, A. C. Rekdal. A Survey of Controlled Experiments in Software Engineering. IEEE Transactions on Software Engineering, vol. 31, no. 9, pp. 733-753 (2005).
6. M.V. Zelkowitz, D. Wallace. Experimental Validation in Software Engineering, Journal of Information and Software Technology, vol. 39, pp. 735-743 (1997).

Aggregation of Empirical Evidence

Marcus Ciolkowski

One of the most important challenges in empirical software engineering today is to better integrate empirical studies with decision support, and to collect appropriate data and experiments. The required steps are to identify the information needed, to collect appropriate studies, and to (objectively) aggregate (i.e., summarize) their results. To be able to make informed decisions on introducing, changing, or evolving technologies and processes in practice as well as research, these decisions have to be based on aggregated trustable (i.e., corroborated) evidence and statements. The benefits of such an approach include reducing the risk of introducing / changing technologies (from industrial point of view), and that it is possible to identify evidence gaps (from research point of view).

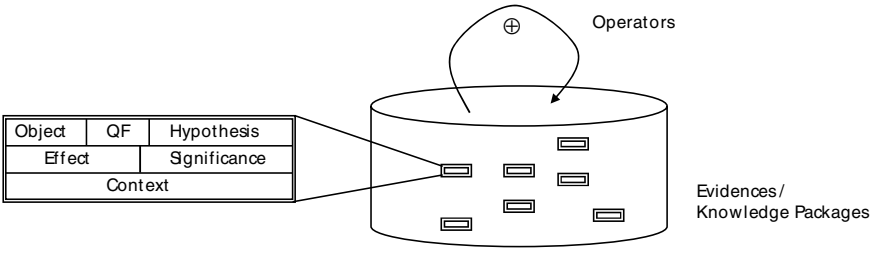
Today, the "state of the art" of aggregation in software engineering is to summarize a set of studies in a tabular form, which is a form of vote counting. Vote counting typically aggregates studies by counting the outcome of significance tests. Intuitively, if a large proportion of studies generate statistically significant results, then the overall effect can be interpreted as being non-zero. Vote counting, however, can be erroneous. In particular, for studies with small effect size and a low number of subjects (i.e., most of the studies in software engineering), vote counting can be strongly biased towards the conclusion that the treatment has no effect. This bias is not reduced as the number of studies increase.

Meta-analysis techniques promise to solve these problems in other fields, such as medicine or psychology. However, attempts at applying these techniques in software engineering have failed so far, as the meta analysis techniques used were not adequate to be applied at current software engineering experiments.

In other words, the problem is that systematic approaches are missing for (1) corroborating statements with (aggregated) trustable evidence, and (2) identifying the need for future studies.

To make progress to solve the stated problem, a framework is needed for goal-oriented aggregation of empirical results, linking empirical results with the goal of aggregation. In other words, the goal is to build corroborated statements from existing evidence.

The framework comprises a model of atomic evidence, knowledge packages (representing aggregated evidence), and aggregation operators to generalize and abstract knowledge.



V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 20, 2007.

Empirical Evaluation in Software Engineering: Role, Strategy, and Limitations

Lionel C. Briand

Though there is a wide agreement that software technologies should be empirically investigated and assessed, software engineering faces a number of specific challenges and we have reached a point where it is time to step back and reflect on them. Technologies evolve fast, there is a wide variety of conditions (including human factors) under which they can possibly be used, and their assessment can be made with respect to a large number of criteria. Furthermore, only limited resources can be dedicated to the evaluation of software technologies as compared to their development. If we take an example, the development and evaluation of the Unified Modeling Language (UML) as an analysis and design representation, major revisions of the standard are proposed every few years, many specialized "profiles" of UML are being developed (e.g., for performance and real-time) and evolved, it can be used within the context of a variety of development methodologies which use different subsets of the standard in various ways, and it can be assessed with respect to its impact on system comprehension, the design decision process, but also code generation, test automation, and many other criteria. Given the above statement and example, important questions logically follow: (1) What can be a realistic role for empirical investigation in software engineering? (2) What strategies should be adopted to get the most out of available resources for empirical research? (3) What does constitute a useful body of empirical evidence?

It is evident that we cannot possibly assess and validate every single software technology being used or adopted under every possible relevant set of conditions with respect to every possible criterion. Empirical studies should therefore (a) target specific technologies which are of economic importance, (b) for which there is significant uncertainty in terms of cost-effectiveness, and (c) which must be investigated under the most representative or plausible conditions. Nevertheless, such assessments will always involve a significant amount of judgment and interpolation. Instead of focusing on unquestionable scientific evidence, our objective is rather to buy information to support decision making. Furthermore, because of the impact of human factors on the cost-effectiveness of many technologies (e.g., education, training, management structure), to be fully understood, the quantitative results of studies must be complemented with qualitative analysis and an investigation of subjective, human perceptions. There are many strategies to do so, ranging from simple questionnaire surveys to think aloud protocols.

An empirical body of evidence in software engineering can therefore be described as a set of studies, each performed under certain explicit conditions, for which both quantitative and qualitative, subjective and objective data have been collected, and based on which certain conclusions and interpretations have been provided. This may be completed by some form of meta-analysis attempting to find an emerging pattern across studies. However, how to make such information reusable in practice?

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 21, 2007.

New Opportunities for Empirical Research

Markku Oivo

Software engineering community has widely recognized the need for more empirical work due to “advocacy research". New methods and tools have been proposed to industry without providing solid empirical evidence to support the claimed benefits. In addition to this need, we suggest new opportunities for empirical research in software engineering and ICT research in general. We propose the Experimental and Explorative Research (EER) strategy to address (1) tool development in software engineering research, and more importantly, (2) application and service development in other ICT areas like research in mobile applications and services. These kinds of projects provide a much more realistic environment for experimentation than traditional student projects, yet they can be controlled much more easily than pure industry projects.

The actual software development work in research projects (not only the use of the tool) is an excellent candidate for empirical research. Tool development process in research projects can be treated as an experiment or quasi-experiment. These projects may include large amounts of professional effort for developing prototype tools and services. Very often the conclusions overlook the development process and are purely related to the end result. They may be similar to an industrial feasibility study and results are product oriented. They ignore development issues, validity concerns are not dealt with, and conclusions mostly lack empirical data from the development process. Influencing factors for success (or failure) of prototype development are not addressed. Generalizability and human factors in the development phase are mostly ignored.

Experimental and Explorative Research (EER) research strategy combines experimental software engineering with exploratory research of new technologies. EER is based on our several years experience of using and developing the approach in research of future mobile applications. In large international projects (e.g. EU projects) explorative application development includes often both industrial software developers and experienced researchers. This kind of an experimental research environment alleviates the subject problems found in student experiments. Furthermore, it does not have the same difficulties found in experimental design and control of industrial projects that are constrained by strict commercial conditions.

EER strategy provides benefits for both worlds: (1) experimental software engineering research benefits from almost industry level projects that can be used as experimentation environments, and (2) future mobile and telecom application research benefits from better control and understanding of the characteristics of the applications and their development methods and processes.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 22, 2007.

Empirical Paradigm: Position Paper

Carolyn B. Seaman

The distinction between "exploratory" and "confirmatory" work is crucial but is too often blurred in current literature. The term "exploratory" refers to empirical work that has as a goal the discovery of new and unforeseen insight. "Confirmatory" research, on the other hand, normally begins with some type of hypothesis or proposition (that has some type of support in previous literature), the confirmation of which is the aim of the study.

My observation in recent literature, however, is that the term "exploratory" is often used as an apologia for a study that is smaller, has less significant results, or is less rigorous than the authors had hoped. This implies that exploratory work has lower standards for size, significance, and rigor. This is not the case. One could argue that the standards for exploratory work should in fact be higher than that for confirmatory. Confirmatory results are rarely interpreted on their own; they are compared to, and interpreted in light of, a body of related work that together sheds light on the validity of a hypothesis. However, exploratory studies are intended to be used to justify further expenditure of resources in the form of future studies. Thus a poorly executed exploratory study potentially has a greater negative impact on the field than a poorly executed confirmatory study. This is not to say that there is no room in our body of literature for reporting exploratory studies that are not perfect. The key is to present our work along with its limitations (which, as an aside, I believe we are getting much better at) and to recognize the contributions of results that we did not expect.

It is my position that qualitative research methods are appropriate for both exploratory and confirmatory research. Further, the most appropriate approach to addressing nearly any research problem in either category is a combination of the two. However, qualitative studies are often confined to the exploratory side of the equation. I would argue that this is due more to the comfort level of reviewers and researchers in our community than on any inherent methodological limitations. There exist guidelines, techniques, and tools from other disciplines for qualitative confirmatory analysis. These would need tailoring and translating to be useful for software engineering and norms would have to emerge about how such studies should be presented. But the largest obstacle to the use of qualitative methods for confirmatory analysis is our own stomach for accepting the idea of hypothesis testing without p-values and tables of statistical summaries. This requires some pioneering examples of rigorous qualitative confirmatory analysis that are both well presented and relevant to our discipline.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 23, 2007.

The Value of Empirical Evidence for Practitioners and Researchers

Austen Rainer

The empirical software engineering research community has two general aims:

1. To understand how software is actually developed and maintained; and
2. To understand what improvements should be made to software development and maintenance, and how those improvements should be implemented.

Empirical software engineering research, therefore, is about both contemplation and action. It is a discipline which attempts to understand phenomena whilst at the same time trying to change those very phenomena (in order to improve them). And it is a discipline that, by definition, promotes empirical evidence as the *primary* source of reliable knowledge for achieving these two general aims.

While the research community promotes empirical evidence as the primary source of reliable knowledge, software practitioners do not seem to value empirical evidence in quite the same way; indeed, software practitioners seem to treat empirical evidence as one among a number of sources of knowledge (others being personal preferences and values, personal experience, the opinions of local experts, and local constraints) where the value of each of these kinds of knowledge varies from situation to situation: for example, in one situation (perhaps a CMM Level 4 company), a decision is made on the basis of empirical evidence; in another situation (a CMM Level 1 company), a decision is made on the basis of local expertise.

As a research community we of course need to (further) strengthen our collaborations with industry, because industry is both the most appropriate source of *empirical* evidence for developing our understanding (the research communities' first general aim) and the intended target for action (the research communities' second general aim). We also need to be aware that industry is also the most appropriate source for these other kinds of knowledge; and that practitioners are not necessarily persuaded to collaborate and to change and to improve primarily because of empirically-based knowledge (particularly quantitative data).

Practitioners' preferences for different kinds of knowledge are implicitly acknowledged in, for example, the design of the Capability Maturity Model (where the lower Levels of the model do not promote the collection and use of quantitative data) and more explicitly acknowledged by, as an alternative example, Evidence Based Software Engineering (derived from Evidence Based Medicine) which seeks to combine various kinds of knowledge to improve practitioners' decision-making about technology adoption.

Given these circumstances, perhaps the most appropriate and helpful recommendations for improvement that the research community can offer to industry are heuristics (rules-of-thumb) based on the careful aggregation of different kinds of empirical and non-empirical evidence.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 24, 2007.

Empirical Paradigm – The Role of Experiments

Barbara Kitchenham

Abstract. This article discusses the role of formal experiments in empirical software engineering. I take the view that the role of experiments has been overemphasised. Laboratory experiments are not representative of industrial software engineering tasks, so do not provide us with a reliable assessment of the effect of our techniques and tools. I suggest we need to concentrate a larger proportion of our research effort on industrial quasi-experiments and case studies. Methodologies for these empirical methods are well-understood in the social science and would appear to be appropriate mechanisms for investigating many software engineering research questions. In addition, I believe we need to make the results of empirical software engineering more visible and relevant to practitioners. To influence practitioners I suggest that we need to produce evidence-based text books and evidence-based software engineering standards.

1 Introduction

In this paper, I discuss the role of formal experiments in empirical software engineering. I believe that we may have over-emphasised the role of formal experiments in empirical software engineering and as a result we have both failed to identify the limitations and risks inherent in software engineering experiments and given insufficient consideration to other empirical methods.

My basic assumption is that the goal of empirical software engineering is to influence the practice of software engineering. This implies that we need empirical methods that provide us with insights into how software engineering works in practice and how changes to the process can result in changes to the outcomes of the process.

In order to explain my concern about formal experiments, I will identify some areas where the nature of software engineering practice is at odds with the requirements of formal experiments and discuss some of the risks that arise because of this. Then I will suggest that quasi-experimental design and case studies might be better suited to some types of empirical study than formal experiments. Finally, I will indicate how we might make the results of empirical studies more visible to practitioners.

2 Software Engineering Practice and Experimental Methodology

Software engineering in practice involves coordinating and integrating many different tasks (analysis, design, coding, testing, quality assurance, project management etc.) that rely heavily on human expertise often in the context of developing innovative products using new technologies. For large scale software engineering, this basic complexity is compounded by the involvement of many different engineers and

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 25 – 32, 2007.

managers working in cooperating teams (sometimes distributed) within one or more industrial cultures. In general it is difficult to identify one task or a single decision and consider its impact in total isolation from its surrounding context.

In comparison, formal experiments abstract tasks away from industrial contexts in order to study in detail specific isolated elements of a process, an event or an artefact. In general the more isolated the object of study is from its environment the easier it is to manipulate and study, but there is a risk that the results will not apply in more complex industrial situations.

Software engineering researchers often debate the use of student subjects, but in my opinion the choice of subjects is far less critical that the selection of materials, tasks and contexts. If our materials are small scale documents with known solutions, our tasks are restricted to those that take less than 2 hours, and the rich industrial context in which software tasks are planned and performed is removed, what is the value of the outcomes of our formal experiments? Clearly there are some cases when we can rely on formal experiments but there are significant risks. We need to consider more that just the scale-up problem, or the student subject problem, for example:

- We may fail to recognise the value of techniques that are not cost effective for small scale tasks but would be valuable for large scale activities (techniques that increase overheads such as documentation, project management or quality assurance would fit this category).
- We may not be able to define realistic *control* situations leading to experimental results that cannot be interpreted by practitioners (e.g. comparing task results based on training people with a new technique with results obtained from people given no training is poor experimental practice; new techniques are best compared with current best practice).
- We may over estimate the impact of our techniques when they are used in controlled situations without the variety inherent in industry practice. This may lead to over-optimistic ROI estimates.
- We may find ourselves examining phenomena that are a result of abstracting the technology away from its usage context not characteristics of the technology itself.
- We may blame practitioners for failure to use our methods when the real problem is our failure to understand the complexity of the context in which our techniques will be used.

If we look at what happens in other human intensive disciplines, we observe that either they are able to perform realistic experiments such as randomised controlled trials in medicine or they use quasi-experimental methods such as those developed by social scientists and educationalists.

The critical property of a randomised controlled trial is that it is a real trial of a treatment (e.g. a new drug or other health care intervention) in a real hospital (or health centre) involving real patients and real doctors, with outcomes that directly affect the health and well-being of the participants. It is extremely rare that we are able to undertake trials of such direct relevance to practitioners in software engineering. In fact I am aware of only one experiment (undertaken by Jørgensen and Carelius [1]) that incorporated a genuine randomised experiment within actual practice. Thus, I do not see software engineering being able to adopt randomised controlled trials as a standard experimental protocol.

It might be argued that we are better served by considering our laboratory experiments to be *exploratory* studies. However, formal experiments were designed with hypothesis testing in mind. It is not clear that they are as well suited to exploratory studies as other empirical methods such as industrial case studies.

Adding a qualitative element to formal experiments does not overcome the objection that they were designed for hypothesis testing, certainly not in the context of laboratory experiments with student subjects. Petticrew and Roberts [2] suggest qualitative research is more appropriate that randomised controlled trials for purposes of *salience* (whether the technology/service matters), *process of service delivery*, *acceptability* (whether the technology/service will be taken up by potential users), *appropriateness* (whether the technology/service is right for the proposed users, *satisfaction* (whether users are satisfied with the technology or service). However, to investigate these issues, researchers would need to obtain the opinion of potential users in a realistic context not surrogate users such as students trying out a small scale task in a laboratory.

I conclude that we should be more ready to perform industrial studies using quasi-experimental designs to support hypothesis testing (or *confirmation*) and qualitative studies (particularly case studies) to support hypothesis generation (or *exploration*). I discuss these approaches in more detail in the next section.

3 Quasi-experiments and Case Studies

The social sciences have developed a large number of quasi-experimental designs for large-scale field experiments, and have a clear understanding of the strengths and weaknesses of these designs. Quasi-experimental designs are designs in which it is impossible to allocate subjects/participants to treatment conditions at random. I suggest that empirical researchers in software engineering need to become more familiar with these types of designs and more open to the opportunities they offer to improve the rigour of large-scale industrial studies.

Quasi-experimental designs began with simple before and after designs which immediately confound treatment effects with the passage of time, but have evolved into far more robust designs. Shadish et al. [3] provide a catalogue of basic quasi-experimental designs incorporating multiple pre- and post-measures and control groups. They also describe designs such as interrupted time-series analysis and regression discontinuity that are almost as rigorous as formal experiments, but have the ability to monitor the impact of large-scale social interventions.

The rigour of quasi-designs has improved as researchers have continued to criticize and improve them. A major element of the criticism will be familiar to most empirical software engineers since it is based on an assessment of study validity. Indeed the validity terms that we use in software engineering have been obtained directly from validity issues associated with quasi-experiments undertaken in education and social policy (not formal experiments). For example, *maturity* validity is particularly important in studies that deal with children since the impact of various social and education programs will be confounded with the children's basic skills increasing as they grow older. Similar a *history* threat is a major problem if families living in poverty that are eligible for one support program (e.g. housing support) may also be

receiving another (e.g. food stamps). However, the studies of validity threats do not end with generic threats applicable to any design but have been refined to identify validity threats specific to particular types of quasi-design. For example, Shaddish et al. [3] provide a detailed list of validity threats for case control studies, and discuss validity issues for other quasi-designs.

3.1 Case Control and Cohort Studies

As examples of fairly common quasi-experimental design consider *case control* studies and *cohort* studies. In *case control* studies, we identify experimental units (e.g. humans, organizations, artifacts) that exhibit some undesirable characteristic (e.g. a project that significantly overruns its budget and timescale). We then match one or more *control*s with each case. The controls are units that do not exhibit the undesirable property but in all other respects match one of the cases. Differences between each case and its control(s) are investigated to look for possible reasons for the undesirable characteristic.

Retrospective case control studies are the standard design used to identify risk factors associated with medical conditions. They would seem an obvious candidate for determining project risk factors. This design has many limitations (see Shaddish et al. [3] Table 4.3 for a complete list). A major problem with such designs is to find the correct characteristics to match the cases and the control. Another problem is that case-control studies are usually backward looking (*retrospective*) studies. Other problems associated with data collection include:

- Underlying cause bias: Project managers of failing projects may reflect about possible causes and thus exhibit different recall than project managers of controls.
- Expectation bias: Observers may systematically err in measuring and recording data so that they concur with prior expectations.
- Exposure suspicion bias: Knowledge of the status (i.e. case or control) may influence the intensity and outcome of a search for exposure to a risk factor.
- Recall bias: Questions about specific exposures may be asked several times of cases and only once of control.

One approach to reducing bias resulting from questioning people about past events is to ensure that interviewers are kept "blind" to case status (i.e. the interviewers who interrogate project staff should not know whether the project was a failure or a success). Although this sounds strange, it is the standard practice for studies that interrogate people about their exposure to medical risk factors.

An alternative design is a forward looking (*prospective*) study. Cohort studies are often prospective. In this type of study we identify a sample of experimental units and observe their progress over time. Medical cohort studies involve millions of subjects over long periods of time (up to 20 years) so this type of design is suitable for large-scale, long-term studies. They are often used to identify the incident rate of diseases in the general population, so they would seem to be appropriate for issues such as the rate of project failures. However, there have been no prospective studies of this type performed in software engineering.

3.2 Evaluating Technology Impact

Two recent studies of the impact of ISO/IEC 15504 (SPICE) have been based on correlation studies ([4], [5]). Correlation studies are observational studies which are weaker methodologically than experiments or quasi-experiments. They always suffer from the problem that they cannot confirm causality. Significant correlations may occur by chance (particularly when a large number of variables are measured), or as a result of a "latent" variable (i.e. an unmeasured variable that affects two measured variables and gives rise to an apparent correlation between the measured variables).

A more reliable approach is to monitor the impact of technology adoption in individual organizations by measuring project achievements before and after adoption utilizing multiple measurement points before and after technology changes. This approach has been adopted by several researchers for CMM evaluations. For example, Dion [6] recorded cost of quality and productivity data for 18 projects, undertaken during a five year process improvement activity. The first two projects were started before the process changes were introduced; the subsequent projects were started as the series of process changes were introduced. Simple plots of the results show an ongoing improvement over time consistent with an ongoing process improvement exercise. However, the provision of data on projects started prior to the process changes gives additional confidence that the effect was due to the process change rather than other factors. Steen [7] provides another endorsement of Dion's methodology. He reviewed 71 experience reports of CMM-based SPI and identified Dion's study as the only believable report of Return on Investment (ROI) of CMM-based SPI.

In another study, McGarry et al. [8] plotted project outcomes before and after the introduction of CMM level 2. The data spanned a 14-year period and included 89 projects. The graphs showed that improvements in productivity and defect rates were not due to the introduction of CMM. The same improvement rate had been observed prior to the introduction of CMM and could be attributed to the general process improvement activities taking place before and during CMM adoption not specifically the adoption of CMM (i.e. McGarry observed a *history* effect). In contrast, improvements in estimating accuracy did appear to be a result of adopting CMM.

3.3 Industrial Case Studies

Quasi-experimental designs allow us to perform quantitative studies investigating factors such as the effectiveness of techniques, or the relative importance of project risk factors. Industrial case studies in contrast allow us to look in detail at the *how* and *why* of software engineering phenomena [9].

It is important to identify what I mean by a *case study*. A case study should be a genuine industrial software engineering project (or project activity), not a toy project, nor a special project performed for the purpose of evaluating a technology or training new staff. All too often researchers use the term case study when they mean *example* (i.e. recreating a previously constructed software artefact using a new technology). In principle, an industrial software project should act as a *host* for a case study. In fact, as Yin points out the most convincing case studies are those that have a strong rationale for case selection. This means that the host project should have

characteristics that make it suitable to address the issues being investigated by the case study. If we are concerned about investigating the way technology works in practice and its impact on practitioners, industrial case studies are likely to be a more reliable methodology than small-scale experiments with an added qualitative element.

4 Visibility of Empirical Software Engineering Results

Several recent publications have made the point that software engineering academics and practitioners trust expert opinion more than objective evidence ([10], [11]). I conclude that that empirical software engineering will not have much relevance to practitioners, if empirical studies have no visibility. For this reason, we need to find a suitable outlet for our results. There are two areas that empirical software engineering should address to make empirical ideas visible to practitioners: text books, which can influence software engineers during their training, and international standards, which are likely to impact industrial practitioners.

We need software engineering text books that incorporate empirical studies to support their discussion of technologies that identify the extent to which technologies have been validated, or under what conditions one technology might be more appropriate than another. Endres and Rombach [12] have made a start at this type of text book, but we need more general software engineering text books that include empirical evidence. Furthermore, text books require summarised evidence not simply references to individual empirical studies, so I we need more systematic literature reviews to provide rigorous summaries of empirical studies ([13], [2]).

We also need evidence-based standards. In my experience the quality of international software engineering standards is woeful. I have no objection to standards related to arbitrary decisions, such as the syntax of a programming language, which are simply a matter of agreement. However, standards that purport to specify best practice are another issue. Software standards of this type often make unsupported claims. For example ISO/IEC 2500 [14] says:

> "The purpose of the SQuaRE set of International Standards is to assist developing and acquiring software products with the specification and evaluation of their products. It establishes criteria for the specification of software product quality requirements, their measurement and evaluation."

I imagine a large number of researchers and practitioners would be surprised to find that the means of specifying, measuring, and evaluating software quality is so well-understood that it can be published in International Standard.

Compare this "International Standard" with the more modestly named "Research-based web design and usability guidelines" [15]. The web-guidelines not only explicitly reference the scientific evidence that supports them; they also define the process by which the guidelines were constructed. Each individual guideline is rated with respect to its importance and the strength of evidence supporting it. The software engineering industry deserves guidelines and standards of the same quality.

5 Conclusions

In this article, I have argued that we have overemphasised the role of formal experiments in empirical software engineering. That is not to say that there is no place for formal experiments in software engineering. Formal experiments can be used for initial studies of technologies such as proof of concept studies. There are also undoubtedly occasions when formal experiments are the most appropriate methodology to study a software engineering phenomenon (for example, performance studies of alternative coding algorithms). However, the nature of industrial software engineering does not match well with the restrictions imposed by formal experiments. We cannot usually perform randomised controlled trials in industrial situations and without randomised controlled trials we cannot assess the actual impact of competing technologies, nor can we assess the context factors that influence outcomes in industrial situations.

To address the limitations of formal experiments, I suggest that empirical software engineering needs to place more emphasis on industrial field studies including case studies and quasi-experiments. In addition, we need to make empirical results more visible to software engineers. I recommend that the empirical software engineering community produce evidence-based text books and campaign for evidence-based standards. Evidence-based text books would prepare future software engineers to expect techniques to be supported by evidence. Evidence-based standards might help practitioners address their day-to-day engineering activities and lead to a culture in which evidence is seen to benefit engineering practice.

Acknowledgements

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. At Keele University, Barbara Kitchenham works on the UK EPSRC EBSE project (EP/C51839X/1).

References

1. Jørgensen, M. and G. J. Carelius. An Empirical Study of Software Project Bidding, IEEE Transactions of Software Engineering 30(12):953--969, 2004.
2. Petticrew, Mark and Roberts, Helen. Systematic Reviews in the Social Sciences. A practical Guide. Blackwell Publishing, 2006.
3. Shaddish, W.R., Cook, T.D., and Campbell, D.T. Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin Company, 2002.
4. El Emam, K. and Birk, A. Validating ISO/IEC 15504 measures of software development process capability. The Journal of Systems and Software, 51, 2000a, pp 119-149.
5. El Emam, K. and Birk, A. Validating ISO/IEC 15504 measures of software Requirements Analysis Process Capability, IEEE Transactions on Software Engineering, 26(6), 2000b, pp 541-566.
6. Dion, Raymond. Process Improvement and the Corporate Balance Sheet. IEEE Software, 10(4), 1993, pp 28-35.

7. Steen, H.U. Reporting framework-based software process improvement. A quantitative and qualitative review of 71 experience reports of CMM-based SPI. Master Thesis, Simula Research Laboratory & Department of Informatics University of Oslo, 29th October 2004.
8. McGarry, F., Burke, S. and Decker, B. Measuring the Impacts Individual process Maturity Attributes have on Software Products. Proceedings Fifth International Software Metrics Symposium, IEEE Computer Society, 1998, pp 52-60.
9. Yin, Robert K. Case Design and Methods. Third edition, Sage Publications, 2004.
10. Barbara Kitchenham, David Budgen, Pearl Breton, Mark Turner, Stuart Charters and Stephen Linkman, Large Scale Software Engineering Questions – Expert Opinion or Empirical Evidence? Experience and Methods from Integrating Evidence-based Software Engineering into Education, Proceedings 4th International Workshop WSESE2003, Fraunhofer IESE-Report No., 068.06/E, 2006
11. Rainer, A., Jagielska, D. and Hall, T. Software Practice versus evidence-based software engineering research. In Proceedings of the Workshop on Realising Evidence-based Software Engineering, ICSE-2005, 2005, http:/cfm.portal.acm.org/dl.
12. Endres, Albert and H. Dieter Rombach. Empirical Software and Systems Engineering: A Handbook of Observations, Laws and Theories, Addison Wesley, 2003.
13. Kitchenham B. (2004). Procedures for Undertaking Systematic Reviews, Joint Technical Report, Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd (0400011T.1)
14. ISO/IEC 25000. International Standard. Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. 2005-08-01.
15. Koyani, S.J., Bailey, R.W. and Nall, J.R. (2003) Research based web design and usability guidelines. National Cancer Institute, Available for download at http://usability.gov/ pdfs/guidelines.html.

The Role of Controlled Experiments in Software Engineering Research

Victor R. Basili

1 The Experimental Discipline in Software Engineering

Empirical studies play an important role in the evolution of the software engineering discipline. They allow us to build a body of knowledge in software engineering that has been supported by observation and empirical evidence. These studies allow us to test out theories, identify important variables and to build models that can be supported by empirical evidence.

But what kinds of empirical studies should we focus on?

There are a variety of study types that can be performed. In many cases the type of study will depend on the circumstances. Much of what we do in the software engineering domain is opportunistic and we are often limited by the situation available.

I will use the word study here to mean the act of discovering something unknown or of testing a hypothesis. Studies can be driven by hypotheses or simply a need to understand. They can use quantitative or qualitative analysis, or a mix of both. They can vary from controlled experiments to quasi-experiments or pre-experimental designs, to simply observations. These latter studies tend to involve more of a qualitative analysis component, including at least some form of interviewing.

The type of study can be an experiment or an observation. On one end of the spectrum, studies can be cause-effect, allowing us to understand the effect of the independent variable on the dependent variable. Sometimes we can do a correlational study, where we can at least recognize that two variables are related. However, sometimes the best we can do is a descriptive study in which we can observe a particular situation. The subject may vary in experience from novice to expert. The experimental setting can be in vivo or in vitro. The type of analysis can be quantitative or qualitative or a mix of both.

The goals of our studies vary from trying to understand the effects of processes and techniques on products to evaluating the product characteristics themselves, from predicting cost and schedule to trading off environmental constraints. For example, when introducing any form of process, method or tool, the organization needs to evaluate its feasibility, usability, and effectiveness in context. These goals help the researcher build knowledge about the discipline as well as help the practitioner understand how to build software systems better.

The varied reasons for study, the immaturity of the discipline in terms of well formulated models, and the opportunistic nature of the circumstances requires a palate of many kinds of studies, types of analyses, settings, experience levels, etc. Each has it benefits and drawbacks and provides insight into our body of knowledge about software engineering. One of our biggest problems is how to combine them to form a coherent body of knowledge.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 33 – 37, 2007.

Approaches vary in cost, level of confidence in the results, insights gained, balance between quantitative/qualitative research, etc. When we are studying a single team the costs are reasonable, we can observe large projects in realistic (in vivo) settings. Our analysis is more qualitative. Examples here include case studies and interviews. If we can study a population over time, we can use quasi-experimental designs. If we can study multiple teams performing the same activity the costs are high, the projects tend to be small, but we can gain statistical confidence in the results. Examples here include fractional factorial designs and pretest /post test control group designs. Thus the ideal is to be able to combine types of study to build the body of knowledge [1].

Thus all forms of study help contribute to knowledge.

2 Use of Controlled Experiments in Empirical Software Engineering

Controlled experiments offer several specific benefits. They allow us to conduct well-defined, focused studies, with the potential for statistically significant results. They allow us to focus on specific variables, measures, and the relationships between them. They help us formulate hypotheses by forcing us to clearly state the question being studied and allow us to maximize the number of questions being asked. Such studies usually result in well defined dependent and independent variables and well-defined hypotheses. They result in the identification of key variables and good proxies for those variables. They allow us to measure the relationships among variables.

So, for example, to identify which of two techniques are best for identifying a certain class of defects, a controlled experiment forces the specification of the treatment techniques, creates training materials.

The very act of defining a controlled experiment forces specification and provides many insights. In running a controlled experiment we are forced to state clearly what questions the investigation is intended to address and how we will address them, even if the study is exploratory. We have to specify context. We have to address the issue of process conformance along with communicable technique definition when we are studying a technique. We can create a design that allows us to maximize the number of questions asked. We can analyze small variations in the variables, such as human ability, experience, learning effects, process conformance, domain understanding, technique definition, etc.

A controlled study provides good insights into why relationships and results do and do not occur. It forces us to analyze the threats to validity, leading to insights in the identification of where replications or alternate studies are needed and where variations might show different effects.

However, a single controlled experiment, outside the context of a larger set of studies, has little value. Controlled studies need to be replicated, varying conditions (to understand the effects in different context), designs (to make up for threats to validity in any one design and context), and allowing for the answering of new and evolving questions from earlier studies. We can combine small focused studies to build knowledge. Each study can be a small contribution to the knowledge tapestry.

In the tapestry of studies it is also important to integrate negative results. Negative results and repeated experiments are important and valuable.

Of course there are drawbacks to controlled experiments. They are expensive. They tend to deal with a microcosm of reality as each study is limited to a specific set of context variables and a specific design. As stated above, you need several such studies where you can vary the context and the design. There are often issues with small sample size, small artifacts analyzed, learning effect issues, unknown underlying distributions, and potentially huge variations in human behavior.

A major threat to validity is the generalization from in vitro to in vivo.

Although a good design has many of the benefits, it is not always easy to achieve. It is easy to miss important points. It is easy to contaminate subjects. TA good controlled experiment usually involves the collaboration of several experimenters and the more people you can get to review your design, the better.

In all forms of empirical study there are problems. Scaling up is difficult and the empirical methods change as the scale grows. It is hard to compare a new technique against the current technique because of the learning curve. You cannot expect perfection or decisive answers. However, insights about context variables alone are valuable.

3 Building a Body of Knowledge

We need to combine studies to build knowledge where each study is a small contribution to the knowledge tapestry. We can build up knowledge by "replication". Replication can take many forms. We can keep the same hypothesis, combining results, we can vary the context variables, e.g., subject experience. We can vary the experimental design, e.g., order of events and activities, study type, artifact size and type. This allows us to balance threats to validity and expand our knowledge.

We can expand our studies by varying the independent variable. For example, we can change the specification or procedure associated with the process, e.g., changing the specificity of the process or technique. We can select another technique from the class of techniques.

We can expand, evolve, change the hypotheses, adding new context variables or evolving the dependent variables.

4 Example Use of Controlled Experiments

We have used controlled experiments in three ways as part of a larger set of studies.

Studying the effects of reading [2, 3, 4, 5]: This is an example of studying the scale up a technique to larger problems. Here we began with a fractional factorial experiment to study the effects of a specific reading technique (reading by stepwise abstraction) versus two specific testing techniques (structural testing with 100% statement coverage and boundary value testing). Based upon the positive results for reading we ran a control group controlled experiment to study the effect of reading by stepwise abstraction on a team development with larger artifacts (Cleanroom study). Based again on positive results, we applied the technique on a real project at NASA Goddard Space flight Center using a quasi-experimental design (case study) to study the scale up of the effects of reading by stepwise abstraction. The next step was to

apply the Cleanroom technique to a series of projects. At each stage we were able to scale up to larger studies using different empirical methods and learn different things about the effects of reading techniques on defect detection and development costs. At each stage we gained the needed confidence in the approach to apply it to larger and larger live projects, perturbing actual developments at Goddard.

Studying different reading techniques [5, 6, 7]: This is an example of replicating a series of controlled experiments, varying the techniques, artifacts, and context. Based upon the studies at NASA using stepwise abstraction as the reading technique, we recognized the need for reading earlier documents than code as well as the need to focus the reading on particular perspectives and defect classes. We ran several studies using Perspective-Based reading. We varied the level of specificity of the techniques, we varied the artifacts being analyzed (requirements, object oriented design documents) and notation of the artifacts (English language, Software Cost Reduction (SCR), notation, UML) being read, and varied the context (students, professionals) including the cultural biases in applying the techniques (U.S, Germany, Norway, Brazil). This allowed us to study and compare the effects of changes to various independent, dependent, and context variables.

Building knowledge about the high end computing development domain [8, 9]: The High Productivity computing Systems project has the to goal of shrinking the time to solution for the development and execution of high end computing systems. In order to better understand software development for HPC systems we are developing and evolving models, hypotheses, and empirical evidence about the bounds and limits of various programming models (e.g., MPI, Open MP, UPC, CAF), the bottlenecks to development, the cost of performance improvement, and the best ways to identify and minimize defects. The idea is to provide some confidence in what works best under what conditions and for whom. The project involves the use of numerous empirical studies with novices and professionals all done in concert, rather than sequentially. Study types include: controlled experiments (grad students), observational studies (professionals, grad students), case studies (class projects, HPC projects in academia), and surveys and interviews (HPC experts). Results are stored in an experience base with chains of evidence connected to hypotheses on one end and implications for various stakeholders on the other end. This mix allows early use of incomplete results rather than the confidence built over a sequential set of studies, each attacking a particular issue.

No particular empirical method is stand alone, but controlled experiments play an important role in building the discipline of software engineering.

References

1. V. Basili, “The Experimental Paradigm in Software Engineering,” in Lecture Notes in Computer Science 706, Experimental Software Engineering Issues: Critical Assessment and Future Directives, H.D. Rombach, V. Basili, and R. Selby, eds., Proceedings of Dagstuhl-Workshop, September 1992, published by Springer-Verlag, 1993.
2. R. Selby, V. Basili, and T. Baker, “Cleanroom Software Development: An Empirical Evaluation,” IEEE Transactions on Software Engineering, vol. 13(9): 1027-1037, September 1987.

3. V. Basili and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, vol. 13(12): 1278-1296, December 1987.
4. V. Basili and S. Green, "Software Process Evolution at the SEL," IEEE Software, vol. 11(4): 58-66, July 1994.
5. V. Basili, "Evolving and Packaging Reading Technologies," Journal of Systems and Software, vol. 38 (1): 3-12, July 1997.
6. V. Basili, S. Green, O. Laitenberger, F. Shull, S. Sorumgaard, and M. Zelkowitz, "The Empirical Investigation of Perspective-Based Reading," Empirical Software Engineering: An International Journal, vol. 1(2): 133-164, October 1996
7. V. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," IEEE Transactions on Software Engineering, vol. 25(4): 456-473, July 1999.
8. S. Asgari, V. Basili, J. Carver, L. Hochstein, J. Hollingsworth, F. Shull, J. Strecker and M. Zelkowitz, "A Family of Empirical Studies on High Performance Software Development," Proceedings of Supercomputing 2004 (SC 04). Pittsburgh, November 2004.
9. L. Hochstein, J. Carver, F. Shull, S. Asgari, V. Basili, J. Hollingsworth, M. Zelkowitz, "Parallel Programmer Productivity: A Case Study of Novice HPC Programmers," Proceedings of Supercomputing 2005 (SC 05), Seattle, WA, November 2005.

Creating Real Value in Software Engineering Experiments

James Miller

Undertaking empirical work to understand the software engineering process is an increasingly common activity. However, due to the nature of most software engineering studies, drawing reliable conclusions from a single study is inherently dangerous. A single study undertaken by a small group of practitioners, in a single location, with a single group of subjects, will possess a large number of parameters, some controlled, some completely unconstrained. Any one of which can potentially cause a significant change in the result of the study; and hence their effect must be controlled, eliminated or understood, if reliable results are to be derived. The normal method of achieving this objective is to repeat the study, often while changing some of the parameters, to see if the original result is stable with regard to repetition and alteration of some of its components. Without this confirming power, empirically based claims in software engineering should be regarded as having at best limited value and at worst with suspicion and mistrust. Hence, despite their now significant volume, software engineering experiments impact has been rather limited upon industrial software processes. The potential problems with single software engineering empirical studies are three-fold: low statistical power; large number of potential covariates with the treatment variable; and the verification of the process and products of the study.

The normal way to tackle these limitations is by replication of the study. Perhaps a more relevant alternate, to the Software Engineering community, is to create a single experiment, which internally contains multiple versions of the experimental theme all based around a common sub-set of hypotheses. This would allow the experiment to amass enough "evidence" to be convincing (if successful) based upon its direct solution of the three aforementioned issues. But while the design of such meta-experiments might seem obvious, the logistical problems associated with their successful implementation are massive. Solutions needing to be found to allow the successful collaboration of such large, and geographically disperse, network of researchers include: (A) the co-ordination of the information flow and experimental design; (B) ensuring suitable knowledge, training and skill levels across all subjects; (C) understanding any, and all, culture issues which may exist between sites; (D) how to obtain funding for such cross jurisdictional projects; and the resolution of clashing ethical, privacy; (E) and general human rights issues between the various sites and jurisdictions.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 38, 2007.

From Silver Bullets to Philosophers' Stones: Who Wants to Be Just an Empiricist?

Guilherme H. Travassos

For a long time scientists have been committed to describe and organize information acquired by observations from the field. To improve the comprehension and testability of the observed information, Bacon's works proposed to organize the way that the experiences should be structured and somehow formalized, starting with the experimental method idea. From that point in time, the ideas regarding experimentation have been explored and evolved into different scientific areas, including physics, agriculture, medicine, engineering and social sciences among others. It has not been different in Software Engineering. By applying the scientific method to organize their experimental studies, software engineers have intensively worked to understand the application and evolution of software processes and technologies. Acquiring knowledge through different categories of experimental studies has supported researchers and practitioners to build a Software Engineering body of knowledge. Families of studies start to be planed and shared among the research community, composing a common research agenda to enlarge such body of knowledge. Based on this, evidence based software engineering is becoming a reality. Nowadays, besides the experimental studies, the experimentation approach represents an important tool to allow the transfer of software technology to the industry and to improve software processes.

Empirical Software Engineering relates to "software engineering based on observation or experience". However, the significance of the word *empirical*[1] does not have enough power to capture the ideas regarding the scientific method (experimentation). By being just *empirical*, software engineers rely on their own experience or myths to observe and report the field. In general, there is no consensus regarding the experimental studies formalization, no common terminology or theory, and replication is not an easy task since a common research agenda can not be identified. In this case, *empirical* sounds like an *ad-hoc* approach, used to explore some particular research interest. It can represent one of the first steps (Aristotelian approach) towards the scientific method, but still far from it.

Our Software engineering community needs to go one step further, looking for ways to better spread and internalize knowledge regarding the scientific method. Observing other more mature areas of science (i.e. physics, medicine, and social sciences) with well defined terminology, theories, studies protocols and common research agenda, *empirical* results has low value. It can also represent a risk for the software engineering scientific area. Software Engineering should not be interpreted as just *empirical* neither observational. It must evolve to be scientifically sounded and, of course, experimental!

[1] *© 2002 Merriam-Webster, Inc.*

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 39, 2007.

Social and Human Aspects of Software Engineering

Helen Sharp

The people-intensive nature of software engineering has been understood for some time. Curtis et al [1] cite a number of studies in a variety of contexts demonstrating "the substantial impact of behavioural (i.e. human and organisational) factors on software productivity", while [2] believe that software companies are particularly vulnerable to people problems. Despite this recognition, however, McDermid and Bennett [3] argue that the neglect of people factors in software engineering research has had a detrimental impact on progress in software engineering.

There is much to be learned from quantitative empirical studies of software engineering that focus on the code and software process artifacts, but we should not focus on these to the detriment of studies that investigate human and social aspects of software engineering. Such studies may be quantitative or qualitative in nature, and they may be field-based or experimental. For example, experimental studies may seek to investigate the effects of different support tools or other work artifacts on an individual's performance or cognitive workload; field studies may seek to understand better the information flows around and within a team of agile software developers.

Conducting such studies and analyzing the results in a meaningful way is likely to require collaboration with researchers from social and cognitive sciences, but I believe that the insights to be gained are substantial.

In my own work I conduct qualitative studies 'in the wild', working with practitioners and real business problems. This approach has its drawbacks, including issues of repeatability and generalisability. One way to address these drawbacks is to perform multiple studies and look for similarities. Another approach is to use these exploratory studies to identify hypotheses that can then be pursued through more focused experimental studies.

My current work focuses on agile software teams (e.g. [5]) and motivation in software engineering; previous work has focused on software quality management systems, and object-oriented development (e.g. 4). In each case, I have stressed the people and their goals rather than the technology they are using.

References

1. Curtis B, Krasner H, Iscoe N (1988) A Field Study of the Software Design Process for Large System, *Comms of the ACM* 31(11), 1268-1287
2. Horvat R.V., Rozeman I., Gyorkos J. (2000) Managing the Complexity of SPI in Small Companies, *Software Process and Improvement Journal,* vol 5, 45-54.
3. McDermid JA, Bennett KH (1999) Software Engineering Research: A Critical Appraisal. *IEE Proceedings On Software Engineering* 146(4):179-186.
4. Sharp, H. and Robinson, H. (2005) Some social factors of software engineering: the maverick, community and technical practices. In Proc. of the 2005 WS HSSE2005, St. Louis, Missouri, May 16 - 16, 2005,. ACM Press, New York, NY, 1-6.
5. Sharp, H. and Robinson, H. (2004) An ethnographic study of XP practices, *Empirical Software Engineering,* 9(4) 353-375.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 40, 2007.

Longitudinal Studies in Evidence-Based Software Engineering

Tracy Hall

Longitudinal studies (LS) generate particularly valuable empirical data. There are many reasons for this, most of which are related to the fact that LS are usually large scale. This allows for a range of rich data to be collected. It also means that the scale of data collected should enable statistically significant results to be generated. Furthermore there are also strong temporal aspects to longitudinal studies. These allow changes over time to be tracked which means that the life of a system can be better understood. It also means that the temporal aspects of process change can be identified. The scale and richness of data, collected over the lifetime of a development project, makes for a valuable empirical investigation.

LS can provide more opportunity for contextual data to be collected. Variation in software development environments means that contextual data is particularly important in software engineering. Collecting rigorous contextual data at the right level of granularity means that research findings are more portable. This allows organizations to customize and adapt findings from empirical research and transfer them into their own projects or environments.

LS are rare in software engineering. Researchers find it difficult to access large-scale industrial software development projects over extended periods of time. Such studies are also expensive and time consuming to run. Consequently many empirical studies in software engineering are either short snapshots of industrial projects or else experiments conducted in laboratories isolated from the industrial context. The efficacy of many snapshot empirical studies is compromised by confounding factors. LS allow a sophisticated understanding of software development to emerge. We argue that data collected in such studies can contribute significantly to the maturation of empirical software engineering research.

Extensive use of LS has been made in medicine, for example Remsberg used LS to identify risk factors for developing cardiovascular disease [1]. The usefulness of LS has also been experienced in a few software engineering research studies. Maximilien and Williams [2] conducted a year-long study with an IBM software development group to examine the efficiency of test-driven development.

LS can produce reliable and comprehensive findings that combine technical and social factors in software development. These findings can be presented in a contextualized form that allows them to be more appropriately transferred to other environments. Such findings can directly contribute to the development of bodies of software engineering evidence.

References

1. Remsberg K, A life span approach to cardiovascular disease risk and aging: The Fels Longitudinal Study. Mechanisms of Aging & Development, 124(3), 2003
2. Maximilien, E. M. and Williams, L. 2003. Assessing test-driven development at IBM. In Proc. of the 25th ICSE, Portland, Oregon, May 03 - 10, 2003, IEEE Computer Society, Washington, DC, 564-569.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 41, 2007.

The Use of Grounded Theory in Empirical Software Engineering

Jeffrey Carver

Empirical software engineering research has much in common with social science research (e.g. Cognitive Science, Psychology). Both types of research focus on understanding how people behave and react in different situations. An important research method for developing hypotheses and insight that is commonly used in these other fields is *grounded theory*. The basic principle behind grounded theory is that the hypotheses and theories emerge bottom-up from the data rather than top-down from existing theory. Using this approach, a researcher begins with an existing data set and abstracts a hypothesis or a theory that accurately describes that data. Then, as more data sets become available, the hypotheses and theories are refined so that they continue to accurately describe all of the extant data.

In empirical software engineering, grounded theory can be very helpful in building a body of knowledge about topics of interest. The grounded theory approach helps researchers develop the right set of questions and hypotheses for a new study or series of studies. This approach can be useful for both the exploratory and confirmatory stages of experimentation. By analyzing the data from existing studies and generating new or refined hypotheses from the bottom up, researches can get an idea both of which hypotheses will be likely to find support in a new study (confirmatory) as well as which hypotheses or research questions have not yet been addressed by the existing data (exploratory). These gaps in the data also provide a useful starting point for planning new studies that will extend the body of knowledge. We have successfully applied this approach to the large body of data on software inspections to generate hypotheses about the impact of an inspector's background and experience on their performance during an inspection.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 42, 2007.

Exploration and Confirmation: An Historical Perspective

Michael S. Mahoney

1 Looking to History

In organizing this session, Dieter Rombach invited me to provide historical perspective on a debate that he anticipated but that did not materialize. Expected to take strong stands on opposite sides of a putative disjunction between exploratory and confirmatory experimentation, the papers by Vic Basili and Barbara Kitchenham instead agreed that, given the nature of the subject and the current state of the field, both are necessary. Before one has something to confirm, one must explore, seeking patterns that might lend themselves to useful measurement. The question was not whether experiments of one sort or the other should be carried out, but rather how experiments of both sorts might be better designed. Nonetheless, some historical perspective might still be useful, not least because discussions of experimentation usually involve assumptions about its historical origins and development. It is part of the foundation myth of modern science that experiment lies at the heart of the "scientific method" created (or, for those of Platonist leaning, discovered) in the seventeenth century. That "method" often serves as a touchstone for efforts to make a scientific discipline of an enterprise and thus forms the basis for much common wisdom about where and how experiment fits into the process.

However, over the past several decades historians and sociologists of science have subjected scientific experiments, both historical and current, to critical study, revealing the complexity and uncertainty that have attended experimentation from the beginning [1]. What tradition has portrayed as straightforward applications of empirical common sense to a readily accessible phenomenal world turn out on closer examination to have involved a delicate interplay of experience, theory, and inspired guesswork, as well as subtle negotiation between the experimenters and the audience they were seeking to persuade. When first carried out and reported, experiments that we now assign to students as canonical examples of scientific method turn out to have been ambiguous in their results and difficult to replicate, and to have provoked disagreement rather than settling it [2].

Moreover, experiment turned out not to have played the role previously attributed to it in the formative period. In "Mathematical vs. Experimental Traditions in the Development of Physical Science" Thomas S. Kuhn sought to break down the monolithic image conveyed by the notion of the Scientific Revolution in the 16th and 17th centuries, a phenomenon also referred to as "the origins of modern science" [3]. The felt need to encompass all the sciences had led historians of science to extend the notion of "revolution" to areas where it was not appropriate, to expand the period well into the 18th century to cover a "delayed revolution" in chemistry, and – perhaps most pernicious – to impose 19th-century categories on a range of empirical and experimental endeavors concerning heat and electricity. Kuhn argued for a more modest view, restricting revolution to a small group of sciences, namely astronomy,

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 43 – 49, 2007.

mechanics, and optics, all of which had mathematical traditions reaching back to classical antiquity. For our purposes, it is only in this area that one can find experiments that might be called confirmatory.

2 Discovering What to Measure

Confirmation presupposes that one knows what to measure, how to measure it, and what the measurements mean [4]. Until one knows enough to isolate a phenomenon by means of a hypothesis that specifies its determinative parameters, one has no choice but to explore its behavior more generally. That exploration may take the form of systematic experimentation, but in many cases it emerges from experience. Experiments in mechanics in the early 17th century provide a good example. Classic among them is Galileo's inclined-plane experiment to confirm the law of falling bodies, to wit, that the distance traversed from rest under natural acceleration varies as the square of the time of fall. However, that experiment rested on the premise that a ball rolling down an inclined plane models the essential quantitative behavior of vertical free fall. To establish that principle, Galileo described a series of experiments with pendulums, showing that, friction and air resistance aside, a bob dropped from a initial height rises again to that same height irrespective of a change of trajectory caused by shortening the length of pendulum by means of a nail on the center line. Behind those experiments lay earlier studies of the bent-arm balance and its relation to the pendulum and the inclined plane. Hence, behind the experiment confirming a quantitative law, and underpinning it, lay a series of exploratory experiments, through which Galileo, guided by considerations from philosophical sources, teased out the parameters of natural acceleration and the means of displaying and measuring their interaction.

In Galileo's case, however, the question is how the study of motion became experimental at all. It had not been up to that time. Galileo presented his results as the second of "two new sciences" (the first was the strength of materials), noting at the start of his exposition that:

We set forth a very new science concerning a very old subject. Perhaps nothing in nature is more ancient than Motion, and volumes neither few nor small have been written by Philosophers about it. Nevertheless, I have discovered several essential properties that are worth knowing but that hitherto have been neither observed nor demonstrated. Some more obvious things have been noted, for example that the natural motion of falling bodies is continually accelerated. But according to what proportion its acceleration occurs has so far not been established; no one, as far as I know, has demonstrated that the distances traversed in equal times by a body falling from rest stand in same relation to one another as do the odd numbers starting from unity. It has been observed that missiles, or projectiles, trace out some sort of curved line, but no one has established that it is a parabola. That it is, and several other things no less worth knowing, have been demonstrated by me, and, what is more important, they open the way to a most broad and excellent science, for which these our labors will be the starting point from which minds sharper than mine will penetrate into the deepest recesses [5].

Although Galileo published his Two New Sciences in 1638, he was reporting on work carried out during the period 1592-1609, when he taught mathematics and related subjects at the University of Padua and acted as engineering consultant to the Arsenal of Venice. Writing the treatise in the form of a dialogue, he set the action in the Arsenal, where, in his opening words, "everyday practice provides speculative minds with a large field for philosophizing." His remark points to a larger background against which the emergence of the new subject and its mathematical and experimental methods must be viewed.

As Galileo proposed, his new science of motion was the starting point for the development of mechanics as the mathematical theory of abstract machines, which culminated in Newton's definition of the subject as "the science of motions resulting from any forces whatsoever, and of the forces required to produce any motions, accurately proposed and demonstrated"[6]. Quite apart from the conceptual issues involved, the very combination of machines, motion, and mathematics brought together subjects that had been traditionally pursued in quite separate realms. Machines were the business of artisans and engineers. Theories of motion were the business of natural philosophers. From Antiquity through the Renaissance, the two groups had nothing to do with one another. Before machines could become the subject of mathematical theory, they had to come to the attention of the philosophers, that is, they had to become part of the philosophical agenda. That process began with the engineers, who during the Renaissance increasingly aspired to learned status, which meant putting their practice on some sort of theoretical basis, that is, expressing what they knew how to do in the form of general principles. In the new literature on machines that appeared over the course of the 16th century, one can see such principles emerging in the form of what I have called elsewhere "maxims of engineering experience" [7]. Though expressed in various ways, they come down to such things as:

```
You can't build a perpetual motion machine.
You can't get more out of a machine than you put into
it; what you gain in force, you give up in distance.
What holds an object at rest is just about enough to
get it moving.
Things, whether solid or liquid, don't go uphill by
themselves.
When you press on water or some other liquid, it pushes
out equally in all directions.
```

Over the course of the 17th century in the work of Galileo, Descartes, Huygens, Newton, and others, these maxims acquired mathematical form, not just as equations expressing laws but also as structures of analytical relations. At the hands of Torricelli, for example, the fourth maxim took the form that two heavy bodies joined together cannot move on their own unless their common center of gravity descends. Combining that principle with Galileo's work on the pendulum and on natural acceleration, Huygens reformulated it as the principle that the center of gravity of a system of bodies will rise to the same height from which it has fallen, irrespective of the individual paths of the bodies. Expressed mathematically, the principle is an early form of the conservation of kinetic energy expressed as mv^2.

The laws of classical mechanics have been tested and retested experimentally since the 17th century, and every student in high school physics is invited to confirm them in a carefully designed laboratory exercise. We do not invite them to look beyond the exercise to appreciate the centuries of exploratory experience in machine building and the construction of buildings and waterworks from which the laws of motion took their start. Nor do we have them read the ancient and medieval philosophical literature that wrestled with the nature and measure of motion. Before Galileo ever started rolling balls down an inclined plane and measuring the times and distances traversed, he pretty much knew what he would find. Indeed, the experiment doesn't make sense except in terms of the expected results [8].

Galileo's own efforts in other areas make the point clear. The laws of natural acceleration constituted the second of two new sciences. The first was the strength of materials, and there Galileo had less accumulated experience on which to draw. His experiments in this realm were much more of a cut-and-try variety, as he explored possible various ways to test an idea that the cohesion of bodies might have something to with nature's resistance of the vacuum. His successors in the Academia del Cimento picked up where he left off, pursuing a range of experimental inquiries without much sense of where they might lead.

3 Discovering What Happens

In drawing out the contrast between the two traditions, Kuhn pointed to the quite different treatments of magnetism and electricity in William Gilbert's De Magnete (On the Magnet, 1600), considered one of the classics of early experimental science. In experimenting on the magnet, Gilbert drew on a large inventory of empirical data provided by earlier experimenters and in particular by mariners, whose experience of the variation and declination of the compass suggested systematic lines of inquiry. By contrast, when Gilbert turned to electrostatically charged bodies, he had no similar body of experience on which to draw. Hence, his experiments were based for the most part on analogies to the properties of magnets. In general, Kuhn noted,

> When [Baconian] practitioners, men like Gilbert, Boyle, and Hooke, performed experiments, they seldom aimed to demonstrate what was already known or to determine a detail required for the extension of existing theory. Rather they wished to see how nature would behave under previously unobserved, often previously nonexistent, circumstances. (43)

17th century experiments on and with the vacuum quite nicely reveal the different patterns. The experiments began with an effort to account for the observed phenomenon that even the best suction pumps could not raise water more than about 30 feet. At first, it was generally attributed to nature's avoidance of a vacuum, and Galileo even proposed an experiment to measure the force of the vacuum. However, by the 1640s people began to understand the phenomenon in terms of the balance between the weight of the column of water and the force of the air pressing on the surface of the water on which it was standing. Such an explanation lent itself to experiments varying the density of the fluid and the pressure of the air, gradually

confirming the hypothesis. The result was the barometer, both as a scientific phenomenon and as a scientific instrument for measuring air pressure.

The experiments involved a closed glass tube, filled with the liquid, and then inverted and placed in an open basin. The fluid would flow out into the basin until the weight of the column and the weight of the air reached equilibrium. Left behind was an empty space at the top of the tube and the question of what, if anything, it contained. Clearly, light passed through it. How about sound? Could an insect fly in it? What would happen to a small animal placed in it? How about a plant? How about chemical reactions? And so on.

The barometer, whether water or mercury, did not lend itself to experiments on these questions, so several people, in particular Otto von Guericke and Robert Hooke (working for Robert Boyle), devised pumps for evacuating the air from a glass receiver or bell jar, making it possible not only to create something close to a vacuum but to vary the air pressure (as measured now by a barometer in the receiver) with some degree of control. For much of the later 17th century, the vacuum became a major site of experimentation, almost all of which was what we would call "exploratory". People placed things in a vacuum to see what happened, and they recorded their observations. But what the observations meant, how they fit with other observations, and how they might be explained, remained open questions. Place a mouse in the vacuum and reduce the pressure. The mouse dies. Why? Air pressure? Something in the air? Something in the mouse? What? How do we find out? Trace the course of any of the experiments in vacuum, and a century or more will pass before empirical exploration gradually gives way to experimental confirmation.

A major hurdle facing the use of experiment to confirm hypotheses was the nature of the hypotheses themselves. The new science of the 17th century rested on the premise that the physical world consisted ultimately of small particles of matter moving according to laws of motion expressible as mathematical relationships. Newton summed it up in Query 31 added to the second edition of his Optics in 1710:

> Have not the small Particles of Bodies certain Powers, Virtues, or Forces, by which they act at a distance, not only upon the Rays of Light for reflecting, refracting, and inflecting them, but also upon one another for producing a great Part of the Phenomena of Nature? For it's well known, that Bodies act one upon another by the Attractions of Gravity, Magnetism, and Electricity; and these Instances show the Tenor and Course of Nature, and make it not improbable but that there may be more attractive Powers than these. For Nature is very consonant and conformable to her Self. How these attractions may be performed, I do not here consider. What I call Attraction may be performed by impulse, or by some other means unknown to me. I use that Word here to signify only in general any Force by which Bodies tend towards one another, whatsoever be the Cause. For we must learn from the Phænomena of Nature what Bodies attract one another, and what are the Laws and Properties of the Attraction, before we enquire the Cause by which the Attraction is performed.

What had worked so well in accounting for the system of the planets, uniting their motion and the motion of bodies on earth under the same laws, should now work at the submicroscopic level. And it should work by experimental means:

> There are therefore Agents in Nature able to make the Particles of Bodies stick together by very strong Attractions. And it is the Business of experimental philosophy to find them out.

As promising as that agenda looks in retrospect, it posed a daunting challenge to researchers at the time, not only to come up with explanations of the requisite mechanical sort but also to devise experiments that confirmed them. Nature proved very hard to grasp at the submicroscopic level, where our senses cannot reach without instruments that depend in turn on an understanding of the phenomena being measured. For the 18th and much of the 19th century, experiments aimed at discovering regularities at the macromechanical level that might give clues to behavior at the micromechanical level, while often purporting to demonstrate the working of hypothetical entities such as the "subtle fluids" that explained the behavior of light (ether), electricity (electrical fluid, later ether), heat (caloric), and chemistry (phlogiston). In short, a lot of exploratory experimentation intervened between Newton's "atomic" chemistry and Dalton's atomic theory of the elements, and much of the science of the 19th century would be directed toward devising models, experiments, and instruments that tied the directly observable world to an underlying reality of matter in motion. Increasingly precise and decisive in confirming mathematical theory, none of that work would have been possible without the exploratory experiments that preceded it. It was a slow process at first, and there is no reason to think that experimental software engineering should follow a different path of development.

Notes and References

1. See, for example, H.M. Collins, Changing Order: Replication and Induction in Scientific Practice (London, 1985), A. Pickering, Constructing Quarks : A Sociological History of Particle Physics (Chicago, 1984), S. Shapin and S. Schaffer, Leviathan and the Airpump: Hobbes, Boyle, and the experimental life (Princeton, 1984), P. Galison, How Experiments End (Chicago, 1987).
2. Owing in part to his casual description, Newton's famous experiments with the prism, in particular the "crucial experiment" showing that monochromatic bands could not be broken down further, did not produce the same results in other people's hands. We tend to overlook how much work goes into preparing student laboratory exercises to make sure that the experiments turn out as we wish them to do in the hands of neophytes. Note too that, when the experiments do not work well, students are invited to explain what they did wrong, not to question the desired result.
3. Reprinted in Thomas S. Kuhn, The Essential Tension: Selected Studies in Scientific Tradition and Change (Chicago, 1977), 31-65.
4. I am here using "measurement" in a broad sense of objective criteria which include qualitative assays, as in the case of chemical reagents and products.

5. Galileo Galilei, Discorsi e dimostrazioni matematiche intorno à due nuove scienze (Leiden, 1638)150, my translation.
6. Isaac Newton, Mathematical Principles of Natural Philosophy, [London, 1687], trans. A. Motte (London, 1728), Preface.
7. M.S. Mahoney, "The Mathematical Realm of Nature", in D.E. Garber et al.(eds.), Cambridge History of Seventeenth-Century Philosophy (Cambridge: Cambridge University Press, 1998), Vol. I, pp. 702-55
8. Several of the experiments for which Galileo is most widely known, such as the dropping of objects from the Tower of Pisa, were never carried out. In most cases, they are aimed at refutation of purported experience, and in some cases they are part of a thought experiment aimed at bringing out logical inconsistency rather than failure to match experience.

Combining Study Designs and Techniques
Working Group Results

Carolyn B. Seaman

1 Background

On Tuesday, June 27, 2006, as part of the "Empirical Paradigm" session of the Dagstuhl Seminar, an ad hoc working group met for approximately 1.5 hours to discuss the topic of combining study designs and techniques. The ad hoc group members were:

Carolyn Seaman, Guilherme Horta Travassos, Marek Leszak, Helen Sharp, Hakan Erdogmus, Tracy Hall, Marcus Ciolkowski, Martin Host, James Miller, Sira Vegas, Mike Mahoney.

This paper attempts to capture and organize the discussion that took place in the ad hoc working group.

2 The Question

The group spent some time at the beginning of the session coming to a consensus on the boundary and meaning of the question we were addressing. We decided that we were to discuss, from an empirical design point of view, the issues involved in designing a group of studies of different types, or a single study employing several empirical research techniques, to address a single research question. We explicitly excluded from the discussion the issue of combining results from different, already completed, studies, i.e. meta-analysis.

3 Motivation

Why is it desirable to combine methods, within or between studies, to address a single research question? One motivation offered had to do with the drawbacks of replication. Replicating a previous study, using the same study design, will often also replicate the flaws in the design of the original study, thus magnifying the effects of those flaws.

Because no study is perfect, using a variety of study designs will diminish the effects of flaws in any one design. Also, different types of studies produce new knowledge at different levels. This knowledge might not be directly comparable, but that's not the point. Their complementarity, not their comparability, is the goal.

Some of the literature in the social sciences talk about three different motivations or goals of mixing study designs: combination, complementarity, and triangulation. These three motivations are closely related and often difficult to separate cleanly.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 50 – 53, 2007.

4 Current State of the Art

We discussed examples in the literature of research designs which incorporated multiple research approaches and techniques, either within one study or among a set of complementary studies. We were able to come up with very few, but we did elicit a set of forms that such studies or sets of studies tend to take. These models are enumerated below.

Model 1: We termed this model the "sequential model". It consists of a linear sequence of studies, all addressing different aspects of a research problem or question, employing a variety of study forms and methods. Often, this takes the form of an iteration between exploratory and confirmatory studies, where the exploratory studies investigate unexpected issues and results that arose from the confirmatory studies, and the confirmatory studies aim to test hypotheses that arise from the exploratory studies. An example of this type of structure is the NASA Software Engineering Laboratory (SEL) Cleanroom studies, which later gave rise to the Perspective-Based Reading (PBR) studies at Maryland.

Model 2: This model is referred to as the "parallel" model. This involves a set of coordinated studies, all addressing the same (at least general) research question, utilizing a variety of (hopefully complementary) research designs. The ongoing VTT research, conducted by Pekka Abrahamsson, on agile methods, was raised as one possible example of this type of study structure. Another example is Lorin Hochstein's recently-completed doctoral dissertation at the University of Maryland.

Model 3: This model consists of a single, central study, which is usually quantitative and confirmatory, with one or more dependent "studies" (or instances of data collection and analysis, even if considered part of the main study), usually qualitative and exploratory. For example, a classroom experiment comparing the effectiveness (in terms of defect detection) of two different reading techniques, might include some post-experiment interviews in order to gather data to help explain the results. Or, a formal hypothesis-testing experiment might be preceded by a focus group in order to help define relevant variables and identify confounding factors that might arise in the experiment. The consensus of the working group was that there are many examples of this in the literature, but in most cases the dependent data and analysis are not reported well. More knowledge could be gained if this other data were collected and reported in as careful a manner as for the central study.

Model 4: The last model, for which we could think of no current examples, consists of a central study that uses current data (i.e. data collected directly from ongoing activities) but also uses historical and benchmark data (collected and analyzed using different methods) to help support and interpret the findings of the central study.

5 Goals

The working group articulated several inter-related goals in this area, although several of them seemed to extend a bit beyond the scope of our discussion. That is, some of the

goals that we felt were necessary for improving the community's ability to effectively combine study types to address research questions were goals that also improved the level of quality of empirical software engineering research across the board.

The goals we articulated are:

1. Better and more explicit connections between published studies. Empirical software engineering researchers need to make better use of the literature by being more familiar with the literature and properly placing their own work in that literature. At a basic level, we all need to read more of each other's work, communicate more about our work to each other, and explain the relationships between different pieces of work in our papers. Ideally, a researcher makes these connections with other work a priori, before designing and conducting a study. This involves thoroughly reviewing the literature and defining an appropriate "gap", then carrying out a study that fills that gap and explaining the literature boundaries of that gap when writing up the study. It also involves identifying other studies whose results can be interpreted in light of the study being reported. We believe this is a short-term goal, and has several concrete sub-goals:
    a. more uniform terminology for empirical software engineering (ESE) that can facilitate the comparison and describing of related studies in papers;
    b. more uniform content structure of ESE papers, i.e. standard templates;
    c. better publication control of ESE papers with respect to making sure that papers include sufficient and thorough related work sections, by training and briefing of conference and journal referees/reviewers.
2. Better contextualization of studies. This may necessitate the definition of a set of standard contextual factors that should be reported for all studies. Although it would not be possible to capture all relevant contextual factors for all studies in a standardized way, an initial set would go along way towards facilitating comparability. Better reporting of context facilitates comparison of studies addressing the same question, even if data is not comparable.
3. Collection of example archetypical method combinations. This would help researchers identify combinations of study designs that work well for particular types of research questions and goals. It would also explain the particular design issues that arise with different combinations. Seaman, 1999, provides a start on such a set, and we look forward to the upcoming (due in early 2007) special issue of the Information and Software Technology Journal, on qualitative studies of software engineering, to address some of these issues, at least for qualitative research.

6 Other Issues

Several issues were discussed by the working group that did not evolve into goals. One is the interplay between exploratory and confirmatory research. The question was raised as to whether or not there is a necessary link between them, i.e. must all confirmatory work be preceded by exploratory work in order to ground the hypotheses being tested? If not, then what criteria should be used to determine whether or not a proposed hypothesis is worth committing the resources to testing? This is considered an open issue, although there was some discussion of the

appropriateness (although often not publishability) of testing a hypothesis based on "common wisdom".

We also briefly discussed simulation, and whether or not it was a valid study design to be considered when envisioning a set of related studies. There are no guidelines for how simulation might fit in, in a synergistic or complementary way, with other types of studies. We discussed an extended taxonomy to classify such studies (i.e., *in virtuo* and *in silico*). Good examples of these types of studies can be found in the ISESE 2005 proceedings, PROSIM and elsewhere. Also, such a taxonomy was introduced at the ISERN meeting in Italy, 2003.

An issue that came up several times is the difficulty of publishing papers that fully address all methodological issues, because of space limitations. Reviewers do not always appreciate all the detail, even if space is not a consideration. More space is needed to, for example, properly contextualize a study, fully explain data from secondary activities (e.g. follow-up interviews), or completely place the study in the body of literature. During the discussion with the plenary group, it was proposed that technical reports are a convenient way to make all the relevant detail available, without putting it all into a published paper. Another possibility that helps in some situations is the use of online sources as supplementary material, but publishers should provide better means for ensuring the persistency of such resources.

Mike Mahoney, the resident historian of science with us at the Dagstuhl seminar, helped our ad hoc group with some context on some of these issues and how they are dealt with in other scientific disciplines. He explained that it is common practice, for example, not to submit full papers prior to conferences, but to submit an abstract, make a presentation of the work at the conference, and write the full paper after the conference, shaped by feedback gained. These papers tend to be longer than ours, and include a lot more "methodology talk". These practices all seemed, to the working group, worth considering in our discipline, although they would require considerable discussion and tailoring.

Finally, the group felt that a cohesive research agenda for empirical software engineering was essential not only for improving the state of the practice of conducting empirical work in general, but also for our topic specifically. That is, there can be much more experience sharing and synergy in terms of combining research designs when more people are following a common agenda and addressing similar problems.

Optimizing Return-On-Investment (ROI) for Empirical Software Engineering Studies
Working Group Results

Lutz Prechelt

1 The Notion of ROI

Return-on-investment (ROI) is a concept from the financial world. In the dynamic view, ROI describes the periodically recurring profits (returns) from fixed financial capital (investment). In the static view, ROI describes the one-time income or saving (return) realized as a consequence of a one-time expenditure (investment). In this case, if the return does not occur within a short time, later parts of the return may be discounted for interest. For our purposes, we will use the static view and ignore discounting. We will call the return *benefit* and the investment *cost*.

The notion of ROI can be generalized to any domain (in particular engineering) in which both cost and benefit can be quantified on a rational scale and both scales use the same units. Most commonly, the scale is either money or time.

ROI can be used for retrospective analysis of the performance of an investment (controlling) or as an aid for decision-making about potential investments (planning).

In the former case, both cost and benefit may be known and the ROI is a single fixed number. Values greater than 1 indicate successful investments, values smaller than 1 indicate failed ones. In the latter case, however, there is usually some uncertainty about the cost and almost always significant uncertainty about the benefit; both expected cost and expected benefit are best described by probability distributions and hence the ROI is also a distribution.

In many cases, the uncertainty can be factored into a base case plus a number of identifiable risks, each of which increase cost or reduce benefit.

2 ROI of Empirical Studies: Industrial View

The view of an industrial organization onto an empirical study in software engineering is well compatible with the definition of ROI as stated above: The study is considered an investment that is made in order to produce a return; the expected ratio of benefit to cost determines whether a suggested study will be performed or not.

When considering how to optimize ROI we may therefore list the kinds of costs and kinds of benefits that typically occur and collect suggestions for improving each of these.

The major kinds of costs are:

- C1. Effort for determining the precise research question.
- C2. Researcher effort for designing the study.
- C3. Researcher effort for performing the study.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 54 – 57, 2007.

- C4. Industrial participant effort for performing the study.
- C5. Researcher effort for evaluating and reporting the study.
- C6. Indirect cost (for instance losses in time-to-market, product quality, or developer motivation).

Note that C2 may involve learning about empirical methods first (in particular for industrial researchers) or learning about the application domain first (in particular for academic researchers).

The major kinds of benefits are:

- B1. Process effectiveness impact: Improved process capabilities with respect to quality or cycle-time.
- B2. Process efficiency impact: Improved process capabilities with respect to cost.
- B3. Morale impact: Improved staff motivation and cooperation, e.g. due to resolution of a dispute.
- B4. Indirect benefit: Improved process capabilities for further process improvement (for instance via further empirical studies).

Note that B1 and B2 apply in particular to technology evaluation studies. They break down into (a) better estimates for the cost and risk of applying the technology, (b) better understanding of its benefits and limitations, (c) initial technology training for subsequent change agents, and (d) improved chance of successfully adapting the technology if it is valuable and rejecting it if it is not.

There may be secondary benefits beyond those listed, such as the publicity if the study results in a scientific publication.

2.1 Optimizing ROI in the Industrial View

We can now use the above partitioning of cost and benefit into elements as a guide for formulating a few general hints how to reduce cost or increase benefit in an empirical study. The various hints work in different ways: each may target a reduction of base cost, an increase of base benefit, the management of a specific class of risk factors, or may just generally reduce the variance of study results (also in the sense of risk management).

- C1. Plan whole research programmes at once rather than individual studies.
- C2. Reuse or adapt the design of a previous study.
- C3. Reuse existing infrastructure (such as instructions, checklists, automation) for data collection.
- C4. Use continuous data evaluation in order to stop using further participants as soon as the study result becomes sufficiently clear; avoid collecting non-essential data; use automation to minimize the work and disruption of the participants; embed the study in the standard software process and adapt your manipulations so as to become indistinguishable from normal project management.
- C5. Use checklists for data evaluation and report in standardized formats.
- C6. Include the empirical study as a risk factor in your project risk management.

- B1. Favor studies whose results can be turned into process improvements easily. Assess the major threats to validity and confounding influences explicitly during the study.
- B2. Ditto.
- B3. Perform empirical studies specifically in areas that have both high relevance and high disagreement; make sure everybody accepts the study design as unbiased. Carefully explain study results and discuss their implications with the stakeholders.
- B4. Introduce explicit experience management.

These hints could be formulated more concretely and in more detail if we discussed different types of research methodology separately (e.g. experiments, surveys, case studies, post-hoc studies, benchmarks).

3 ROI of Empirical Studies: Scientific View

From a scientific point of view, most of what has been said above about cost does apply, however the benefit structure is completely different.

The major kinds of benefits from a scientific point of view are:

- B1. The specific, immediate insight gained from the study.
- B2. New conjectures that help in formulating preliminary theories and corresponding research programs.
- B3. Impact on theory: Partial validation or refutation of a previous conjecture or theory.
- B4. Impact on research agenda: Improved ideas and empirical capabilities for future studies.
- B5. Indirect benefit: Improved trust with industrial partners (or at least an extended track record) and thus increased likelihood of opportunities for empirical studies.

The above formulations use the term *theory* in a rather broad sense; most statements about mechanisms and relationships that are somewhat abstract qualify as theories.

3.1 Optimizing ROI in the Scientific View

- B1. Make sure the study design provides insight even if the expected results do not occur.
- B2. Observe sufficiently many different variables or phenomena at once so that you will probably be able to formulate a new conjecture. In particular, do not unnecessarily restrict the study to quantitative observation only.
- B3. Always formulate some kind of expectation, even if the study is mostly exploratory.
- B4. Perform a post-mortem for an empirical study much like you should do for a software project; keep good ideas and identify mistakes.
- B5. Work on human relationships along with working on the study.

4 Conclusion

While the concept of ROI can be directly applied to individual empirical software engineering studies if an entirely industrial view (focusing on cost efficiency) is used, it should be obvious that ROI in a strict sense cannot be applied to scientific considerations of research benefits, as this would require assigning a monetary value to insights (and do so before you even had them!), which does rarely make any sense.

Nevertheless, the above partitioning of cost and benefit into components and subsequent reflection on ways for optimizing each component suggest that ROI considerations can provide useful guidelines for optimizing research efficiency. Concrete examples for such optimizations can be found in many published studies and are waiting to be reused.

Acknowledgements

Thanks go to the members of the breakout group whose work provided input for this writeup: Frank Houdek, Barbara Kitchenham, Jürgen Münch, Dietmar Pfahl, Austen Rainer, and Laurie Williams. Barbara, Dietmar, Austen, Laurie, and Giovanni Cantone helped significantly in shaping this summary.

The Role of Controlled Experiments
Working Group Results

Andreas Jedlitschka and Lionel C. Briand

Abstract. The purpose of this working group was to identify which role controlled experiments play in the field of empirical Software Engineering. The discussions resulted in a list of motivational factors, challenges, and improvements suggestions. The main outcome is that, although the empirical Software Engineering community, over the last 14 years, has matured with regard to doing controlled experiments, there is still room for improvement. By now the community has understood under which conditions it is possible to empirically evaluate Software Engineering methods, techniques, and tools, but the way controlled experiments are designed, performed, and reported still lacks the level of quality of other disciplines such as social sciences or medicine. The generalizability of the results of controlled experiments is one major concern. Furthermore, more emphasis should be put on the role of empirical Software Engineering education.

1 Introduction

The working group on "The Role of Controlled Experiments" consisted of (in alphabetic order): Lionel Briand (chair), Giovanni Cantone, Jeffrey Carver, Tore Dyba, Andreas Jedlitschka, Natalia Juristo, Sandro Morasca, Nachiappan Nagappan, Markku Oivo, and Elaine Weyuker.

Working method: The chair introduced the points that were to be addressed by the participants. Each participant was asked to give a short statement with regard to the topic at hand, by preferably raising points that had not been addressed before.
Topics of discussion:

- The first round was dedicated to the motivations for doing controlled experiments, which not only imply artificial settings but very often student as subjects.
- During the second round, participants were asked to reflect about the challenges faced during controlled experiments, e.g., difficulties experienced while conducting an experiment or while analyzing its results. Improvement suggestions were also debated while reflecting on these challenges.

1.1 Working Definition of Controlled Experiments

The first thing to be clarified was to agree upon a working definition of the main characteristics of controlled experiments (CE) in software engineering. They usually imply the following:

- Small samples
- Random assignment (but not necessarily random sampling) to treatments

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 58 – 62, 2007.

- Control of extraneous factors (blocking, counterbalancing, etc.)
- Artificial, lab settings
- Well-defined (but small) tasks and (artificial) artifacts
- Proper training for the tasks at hand
- Student or professional subjects

A summary of the working group results was presented during the discussion and summary session by the chair. The comments made on the topics throughout the workshop are summarized in the following sections.

2 Motivation

The working group discussed the motivations for the use of CE in software engineering and came up with the following consolidated list:

- CE are used to investigate/understand cause-effect relationships.
- CE are a good mechanism to gather initial evidence about such relationships.
- CE are also useful for feasibility studies as a preparatory step to field studies. The argument is that, in many cases, if an approach does not work in laboratory settings, it will likely be unsuccessful in more realistic settings. In addition, CE help to identify contextual factors, better control assignments to treatments, and refine measurement before a more extensive and expensive field study.
- CE can also be used to convince industry to invest in larger field studies by showing promising initial results.
- CE give more control over extraneous variables, by using measures like blocking e.g., with regard to the subjects' level of expertise.
- CE can be used as an education mechanism, showing students that it is possible to empirically validate software technologies and using experiments to convince them of innovative technologies.
- CE are necessary in case a technique is not yet used in practice and/or if the training of practitioners would be prohibitive.
- For certain kinds of questions, CE are a better trade-off in terms of threats to validity.
- CE tend to be easier to replicate as they are more controlled.

As a side effect, the discussion also raised some points with regard to using students as subjects in CE.

- Pro students:
  - For certain tasks students are much better trained than most practitioners.
  - They are much less expensive than practitioners.
- Contra students:
  - They do not know anything about the real world of software engineering.
  - They do not have comparable experience.

Comment from the authors: There are several papers that discuss the pros and cons of using students as subjects [1], [3], [5]. There is no common agreement on this point.

Nevertheless, it is accepted that CE in a real industry setting are very challenging in terms of cost and time (for all parties involved), though few examples have been reported.

3 Challenges and Difficulties

The working group also discussed common difficulties encountered while performing CE or while using CE results to perform further research:

- The average quality of the reporting of CE results is considered to be low in software engineering. In contrast to other disciplines, there is no commonly agreed standard way of reporting results [4], though we can probably get good guidelines from other experimental fields.
  - Crucial information is often missing, not only for practical use but also for researchers to build on existing results, e.g., context information (such as, skills and experience of subjects).
- The context of CE is often not realistic in many ways:
  - The long term effect of the treatment is not measured,
  - Typical project pressure is missing, e.g., through multiple parallel projects,
  - Interruptions, e.g., by customers, cannot be simulated,
  - There are no tight time constraints as in a typical project context,
  - There is no turnover like on regular project
- Though they are crucial, negative results (the null hypothesis cannot be rejected) are often not reported. They tend not to be accepted by journals or conferences and researchers are too often reluctant to provide evidence that there are no benefits for their novel approaches.
- Conformance to treatment: a certain level of lack of conformance is unavoidable, but should be minimized. Therefore, we need a mechanism to detect lack of conformance so that we can account for it during data analysis. Conformance of subjects to treatment is often far from perfect. In other words, it is often the case that subjects do not conform perfectly to the procedures they are prescribed to follow. This is hard to measure, but must be taken into account while performing the data analysis.
- The experimenters can unduly influence results. There were several comments that such kind of distortion has to be reduced to the maximum extent possible.
- CE are not suitable for certain research questions, e.g., questions that aim at measuring the long-term effects of a technology.
- It also seems that in past software engineering research, controlled experiments were also probably used for questions that do not lend themselves to such an experimental strategy – a key, but difficult, issue is to decide on the appropriate strategy (research method) to answer a given question.
- Due to small samples, there is usually low statistical power. This is particularly true if the effect size is small. It was reported [2] that the average power of detecting a medium-sized effect size (according to Cohen's definition) was found to be 0.36. Thus the chance of detecting phenomena with medium effect sizes is

around one in three. The general conclusion was that the statistical power of software engineering experiments falls substantially below accepted norms.

- Replications: We face difficulties when replicating experiments that sometimes result in unintentional changes, and furthermore, intentional changes are unavoidable, as replications are usually performed under different constraints and with different objectives in mind. As show in recent studies [7], replications tend to confirm the original findings when they are conducted by the original authors, while the opposite seems to be the case when the replications are performed by other authors. Furthermore, replications tend to change several aspects of the original study (e.g., training, material). Though such changes have to be expected, if not carefully planned or unintentional, such changes can make any interpretation of differences in results difficult. Furthermore, it was mentioned that experimental packages are not that clear, e.g., regarding the context.
- Generalization of or extrapolation from CE is often difficult and only suitable for certain questions. In particular, this argument was raised with regard to the usage of students as subjects and the artificial setting of CE. Even CE in industrial contexts present difficulties, as the subjects involved are often not representative and the project conditions are not realistic.

4 Improvement and Suggestions

The following suggestions for improvement arose from the discussions:

- The generalization issue can be alleviated through qualitative analysis (to understand why and how the effect takes place) [6], a focus on potential effect instead of average effect, and a solid theory in the context of which to interpret results.
- Controlled experiments should be used as low-risk pilot studies to prepare larger studies.
- Any experiment design should include a mechanism to detect subjects' deviations from prescribed experiment procedures, e.g., whether they actually used the techniques they were supposed to use to perform a task. This can then be used as a factor in the subsequent data analysis.
- More effort should be spent on increasing the number of industrial case studies, as there is clearly an imbalance in the current research literature, which focuses mostly on controlled experiments in artificial settings.

5 Conclusions

The group discussions converged towards a consensus. Controlled experiments are a useful means for investigating cause-effect relationships, collecting initial evidence, and performing pilot studies. However, generalization is a difficult issue and interpretation always has some level of subjective judgment. It is also of the outmost importance to interpret results in the context of well-defined theories. But theories must also be grounded in experience, and in order to build theories, it would be necessary, though costly, to plan families of experiments, as no single experiment can

yield a generalizable result. Furthermore, the design and reporting of controlled experiments has yet to improve in software engineering. For example, we need the design to account for the subjects' possible lack of conformance to prescribed procedures and report results in a way that enables future meta-analysis and replications. Software engineering must develop specific procedures for designing and reporting controlled experiments in a way that fits its needs.

Acknowledgements

This report is the result of a group work. The authors would like to thank all working group participants, namely Lionel Briand, Giovanni Cantone, Jeffrey Carver, Tore Dyba, Andreas Jedlitschka, Natalia Juristo, Sandro Morasca, Nachiappan Nagappan, Markku Oivo, and Elaine Weyuker.

References

1. Carver, J., Jaccheri, L., Morasca, S., and Shull, F. "Issues Using Students in Empirical Studies in Software Engineering Education." Proceedings of 2003 International Symposium on Software Metrics (METRICS 2003). September, 2003. p. 239-249.
2. Dybå, T.; Kampenes, V.B., and Sjøberg, D.:"A Systematic Review of Statistical Power in Software Engineering Experiments," Information and Software Technology, Vol. 48, No. 8, August 2006, pp. 745-755.
3. Höst, M.; Regnell, B.; Wohlin, C.: Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment, Empirical Software Engineering, 5, 3, 11/1/2000, Pages 201-214
4. Jedlitschka, A.; Pfahl, D.; Reporting Guidelines for Controlled Experiments in Software Engineering; In Proc. of ACM/IEEE Intern. Symposium on Software Engineering 2005 (ISESE2005), Noosa Heads, Australia, Nov 2005, IEEE CS, 2005, pp. 95-104
5. Kitchenham, B.A.; Pfleeger, S.L.; Pickard, L.M.; Jones, P.W.; Hoaglin, D.C.; El Emam, K.; Rosenberg, J.: Preliminary guidelines for empirical research in software engineering; IEEE Transactions on Software Engineering, Vol. 28, No. 8 , Aug 2002, pp. 721 -734.
6. Carolyn B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," IEEE Transactions on Software Engineering ,vol. 25, no. 4, pp. 557-572, July/August, 1999.
7. Sjøberg, D.; Hannay, J.E.; Hansen, O.; Kampenes, V.B.; Karahasanovic A.; Liborg N.-K.; Rekdal, A.C.: "A Survey of Controlled Experiments in Software Engineering," IEEE Transactions on Software Engineering, Vol. 31, No. 9, September 2004, pp. 733-753.

The Empirical Paradigm
Discussion and Summary

Marcus Ciolkowski, Barbara Kitchenham, and Dieter Rombach

1 Introduction

The session was structured into fours parts:

1. Two presentations, which addressed the use of empirical validation in software engineering.
2. Two presentations about exploratory versus confirmatory experiments.
3. A working session, which addressed questions that arose from the presentations and which involved splitting into three working groups to address different issues followed by a plenary feedback session.
4. A wrap-up by Mike Mahoney from a historian's perspective.

2 Approaches for Empirical Validation

Marvin Zelkowitz and Walter Tichy provided the introductory papers for this session. They both presented reviews of published papers with the aim of identifying trends in the extent and nature of empirical validation in software engineering.

Marvin Zelkowitz presented an update of a previous study undertaken with Dolores Wallace in 1998, which classified papers in ICSE, TSE, and IEEE Software according to the type of validation (if any) reported in the paper. Compared with the previous study, papers written between 1995 and 2005 contained more validation. There were more case studies, dynamic analysis studies and use of data repositories, the later two findings being related to the analysis of open source software. The level of controlled experiments remained stable but low at 7%.

Walter Tichy reviewed papers in the Journal on Empirical Software Engineering over the 10½ years of its existence. He used a classification scheme based on psychological research. He found that studies concentrated on metrics, software process, inspections, and project planning / estimation. Topics such as programming languages and programming paradigm were not addressed. Overall, he found more professional subjects than students, but students were used more often than professionals in experiments. He noted that 26% of experiments are replications.

2.1 Discussion

Following these presentations, the workshop participants discussed the issues they raised for empirical software engineering. Issues and questions included:

- What are really important empirical results? We are in competition with other fields for research funding, and also in competition for research students. What

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 63 – 67, 2007.

have we achieved in empirical software engineering that would appear interesting to students?
- We still have a limited understanding of empirical studies in other fields, and a naïve view of what it means to do empirical studies.
- It takes a lot of expertise to scope questions for investigation. Often, people with an ESE background are not experts in the domain, and vice versa. A good research team needs empirical expertise and domain expertise.

Further discussion identified a list of important issues for empirical software engineering:

- How can we increase the acceptance rate for ESE papers?
- How can we assure the quality of validations?
- How can we increase collaboration with other CS/IS fields and generally increase the involvement of ESE researchers in the CS community at large?
- How can we come up with more realistic studies?
- What would be the most interesting topics/domains for the next five years?
- Can we apply ROI to empirical research?
- Can we learn from best studies and identify why they were successful?
- Should we perform root-cause analysis of why ESE is not valued; what can we do, as a community, to increase the reputation of ESE work?
- How do we stand with regard to CS?
- Do we need to have benchmarks?
- Are there constructive ways to increase the reputation of ESE?
- How can simulation contribute to empirical software engineering?
- We are in the business of buying information for decision making; we need realistic expectations of the outcomes of ESE.
- We need more synergy between domains.

3 Exploration Versus Confirmation

Empirical studies range from exploratory studies of new, badly understood software engineering approaches for the purpose of incremental learning, to confirmatory studies of the properties of well-defined software engineering approaches. Victor Basili and Barbara Kitchenham presented their views on the use of controlled experiments for these types of study, addressing the following questions:

- Which of these study forms is appropriate under what circumstances?
- How do we deal with validity threads (especially for exploratory studies)?

Victor Basili pointed out the importance of controlled experiments in understanding causal relationships. He pointed out the need to learn how to combine different types of studies and presented examples of reading experiments and clean room experiments. Using controlled experiments, we can formulate hypotheses, study variations in context factors, improve and refine tools, and identify and study variations of context (explaining the results of replications including negative results).

However, he acknowledged that generalization was a major problem and proposed generalization by means of a "family tree" of related experiments.

Barbara Kitchenham took the view that empirical software engineering had concentrated too much effort on controlled experiments, which, by their very nature, were very different from industrial software engineering practice. She advocated more industrial studies using case study methods and quasi-experimental designs. She also suggested that we need to change the views of industry and government about the importance of objective evidence by supporting evidence-based standards and evidence-based text books.

3.1 Discussion

The discussion showed that the term "confirmation" may be misleading, and that the term "hypothesis testing" should be considered instead. Other issues included:

- How can we capture context and related work in order to better position our own research?
- How do we ensure the quality of empirical evaluations?
- We need to capture context-related information from empirical studies more systematically.

4 Working Groups

Based on the discussions in the previous sessions, the participants decided to form three working groups: one that addressed the questions of how to combine study designs, one on how to assess the ROI of empirical studies, and one that aimed at defining the role of controlled experiments within ESE.

Carolyn Seaman chaired the working group on Combining Study Designs and Techniques. The group identified four different models of research designs that incorporate multiple research domains:

1. The *sequential* model, which is a linear sequence of studies addressing different aspects of the same research problem or question.
2. The *parallel* model, where different coordinated studies address the same research question, each using different research designs.
3. The single central study (confirmatory) with dependent studies (exploratory).
4. The central study, which uses current data with historical and benchmark data to help interpret findings.

The working group identified several goals in this area:

- Better and more explicit connections between published studies (short-term).
- Better and more formal explication of secondary data collection and analysis.
- Better contextualization ("standard" context factors).
- Collections of good examples of method combinations.

Lutz Prechelt chaired the working group on ROI. Costs are incurred while finding the research question, performing the study and disseminating the results. Combining study designs and techniques is an important topic to improve the ROI of studies.

The working group considered the benefits of ESE:

- Valuable insights come from summaries / reviews;
- Counter-intuitive results were perceived more valuable; because often, we have beliefs but no evidence. However, evidence that confirms beliefs is not considered as valuable.
- One problem is that even when evidence is available, this does not necessarily imply any impact in practice (see inspections/reviews).

It was noted that risk is also an important aspect of ROI assessment that the working group had not considered.

Lionel Briand chaired the working group on the Role of Controlled Experiments. They agreed on a definition of controlled experiments: controlled manipulation of independent variables to study the effect on dependent variables while keeping all other influences constant; one technique to achieve this is randomization.

They noted a number of challenges:

- Generalization, assuring conformance to treatment.
- Controlled experiments are more suited to evaluating the potential of a technique, not to evaluate an average effect.
- Lack of power, small size effect in controlled experiments.

One problem is that controlled experiments are sometimes used to answer research questions where they are not appropriate. In general, more published case studies are needed.

- There are many industrial case studies, but industrial people are not encouraged to publish / write them up; so this knowledge is lost.
- Other important issues are the embarrassment factor (i.e., hiding bad results) or sensitivity.

5 Exploring and Confirming – A Historian's Perspective

Mike Mahoney, a science historian from Princeton, provided us with a historical review on how other disciplines have dealt with the issue of "exploratory versus confirmatory studies".

In particular, he considered how engineering disciplines emerged from theoretical sciences when practical applications could be envisioned. For example, machines became part of the philosophical agenda in the 15^{th} / 16^{th} century, which was quite a revolution to medieval views. Material sciences emerged from engineering with the need to scale up to large scales.

Engineering problems led to "real-object" experimentation, for example, magnetism vs. static electricity; and many of the original studies were exploratory. Replication was often not possible (the machinery was too expensive, and/or too great a level of expertise was required).

ESE should not feel that we are behind, but this is normal progress in the maturing of science; we still need to probe / explore what the important factors are in software engineering. Much of science rests on engineering experience (we can build before

we understand) and we need to accept that wrong models can be useful to advance science (and useful for predictions in practice).

6 Discussion and Summary

Experiments allow us to investigate cause-effect relationships. However, this is partly a social process, which involves us as a community accepting certain results.

Generalization remains a major issue for controlled experiments undertaken under laboratory conditions. We need to be aware that we are not interested in how something works on a particular subject, but on a larger population. The question is how can we extract / generalize from formal experiments to industrial settings? We need to both combine studies properly and be able to assess the validity of our studies.

Generalization can be approached by building models. Biologists used to build models from data gathered during experiments. However, building models through controlled experiments is dangerous, as this can be misleading and lead to false models ("laboratory world"). Biologists initially followed this approach, then they reversed the approach. Now, they first create a logical model, test that in the laboratory, and later in the real world.

Other questions that ESE needs to address include:

- How to combine results effectively (i.e., meta analysis) in order to build an accepted body of knowledge.
- How to combine the different kinds of studies.
- What the role of controlled experiments is within ESE.
- How we can develop testable models and hypotheses.

Measurement and Model Building
Introduction

Victor R. Basili

Abstract. The goal of empirical study is to build, test, and evolve models of a discipline. This requires studying the variables of interest in multiple contexts and building a set of models that can be evolved, discredited, or used with confidence. This implies we need to perform multiple studies, both replicating as closely as possible and varying some of the variables to test the robustness of the current model. It involves running many studies in different environments, addressing as many context variables as possible and either building well parameterized models or families of models that are valid under different conditions. This is beyond the scope of an individual research group. Thus it involves two obvious problems: how do we share data and artifacts across multiple research groups and what are good methods for effectively interpreting data, especially across multiple studies.

1 Data Sharing

How do we share and combine data from multiple studies, given the issues like the protection of intellectual property, proprietary corporate data, data ownership, etc. This leads to questions like:

- How should data, testbeds, and artifacts, be shared?
- What limits should be placed? What credit should be given? How does one limit potential misuse?
- What is the appropriate way to give credit to the organization that spent the time and money collecting the data, developing the testbed, building the artifact?
- Once shared, who owns the combined asset and how should it be shared?
- How are the data and artifacts maintained? Who pays for it?

At this base level, guidelines need to be followed to make sure there is access to all kinds of data and the laboratory manuals used to record how that data was collected.

2 Effective Data Interpretation

Once we have performed a study, run the experiment and saved the quantitative results in a data base of some form or performed a structured interview as part of the case study, we need to perform effective analysis and interpret the data in the context in which it was collected and against other contexts where data has been collected.

Interpreting the results of a study in the larger context is complex enough, but the idea of combining the results from multiple studies, is a really difficult one. This is a

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 68 – 69, 2007.

fact even when the studies are of the same type, e.g., two controlled quantitative experiments, because the context is almost always different. It is more difficult when the studies are of different types, e.g., a quantitative experiment and qualitative case study.

When we find agreement how much can we generalize, how do we incorporate the context variables in the interpretation, how do we assign the degree of confidence in the interpretation? When we find disagreement, do we expand the model, identify two different contexts, or reject the model?

When we add this to the problem that the studies may have been done by different research groups, the problems are multiplied.

We need guidelines that support the sharing of data and data interpretation methods that allow us to combine the results of multiple studies. The former is a matter of getting community agreement and support for maintainable data bases. The latter requires more research into better interpretation methods.

Data Collection, Analysis, and Sharing Strategies for Enabling Software Measurement and Model Building

Richard W. Selby

Abstract. Successful software measurement and model building require effective strategies for data collection, analysis, and sharing. We summarize a template for data sharing arrangements, apply this template to two data collection environments, and illustrate the resulting data analyses using actual empirical studies.

1 Introduction

Current and future technological systems are increasingly software-intensive and large-scale in terms of system size, functionality breadth, component maturity, and supplier heterogeneity [Boe95]. Organizations that tackle these large-scale systems and attempt to achieve ambitious goals often deliver incomplete capabilities, produce inflexible designs, reveal poor progress visibility, and consume unfavorable schedule durations. Successful management of these systems requires the ability to learn from past performance, understand current challenges and opportunities, and develop plans for the future. Effective management planning, decision-making, and learning processes rely on a spectrum of data, information, and knowledge to be successful. However, many organizations and projects possess insufficient or poorly organized data collection and analysis mechanisms that result in limited, inaccurate, or untimely feedback to managers and developers. Organization and project performance suffers because managers and developers do not have the data they need or do not exploit the data available to yield useful information.

Measurement-driven models provide a unifying mechanism for understanding, evaluating, and predicting the development, management, and economics of large-scale systems and processes. Measurement-driven models enable interactive graphical displays of complex information and support flexible capabilities for user customizability and extensibility. Data displays based on measurement-driven models increase visibility into large-scale systems and provide feedback to organizations and projects. Successful software measurement and model building requires effective strategies for data collection, analysis, and sharing. We summarize a template for data sharing arrangements, apply this template to two data collection environments, and illustrate the resulting data analyses using actual empirical studies.

2 Template for Data Collection and Sharing Agreements

A draft working paper [Bas06] outlines properties of shared software artifacts:

- permission,
- credit,

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 70 – 76, 2007.

- feedback,
- protection,
- collaboration, and
- maintenance.

This paper also organizes these properties into a "licensing structure" that has the following elements:

- lifetime,
- data,
- transfer to third party,
- publication,
- help,
- costs, and
- derivatives.

Some foundational issues regarding data organization, attribution, and distribution are outlined as follows. For a data organization approach, what data formats facilitate data sharing and useful interpretation? Research groups desire a concise summary of data source, data collection method, data validation approach, etc. This concise summary includes descriptions of observations and attributes/metrics for each observation and incorporates any limitations or cautions on data usage. A basic data sharing format uses standard tables of data organized in columns and rows where, for example, each row is an observation ("data point") such as a project, component, process, etc. and each column is an attribute (symbolic or numeric "metric") for the observations. An example environment that uses this data format is the University of Maryland Software Engineering Laboratory (SEL) that has data descriptions and a set of tables with one table for component metric data, one table for effort data, one table for change/defect data, etc. For a data attribution approach, what attribution approaches facilitate data sharing and useful interpretation? A useful approach is to have one standard acknowledgement for each data set and one or more standard citations. An example environment that uses this data attribution approach is the University of Southern California Center for Software Engineering (CSE) that has data sharing and standard acknowledgement agreements among its COCOMO 2.0 industrial affiliates. For a data distribution approach, what approaches for data distribution facilitate data sharing and useful interpretation? A useful approach is to have a written agreement or license, and these arrangements have many variations, including fees, in-kind contributions, unrestricted rights, restricted rights (each user needs "approval"), access for data contributors only, access for researchers only, etc. There are many examples and analogies of data distribution approaches including public domain, open source, license without fee, license with fee, intellectual property model (such as protection/royalty for some period), etc.

3 Example Data Collection and Sharing Environment: Software Engineering Laboratory (SEL)

The SEL environment contains over 25 projects in the problem domain of ground support software for unmanned spacecraft control. The projects range in size from a

few thousand source lines to over several hundred thousand source lines. There are tens to hundreds of software modules in each project. The projects vary in functionality, but the overall problem domain has an established set of algorithms and processing methods. Software personnel at the NASA Goddard Space Flight Center in Greenbelt, Maryland, U.S.A. record information about their development processes and products into repositories on an ongoing basis using a set of data collection forms and tools [Sof83]. Data collection occurs during project startup, continues on a daily and weekly basis throughout development, captures project completion information, and incorporates any post-delivery changes. For example, effort, fault, and change data are collected using manual forms while source code versions are analyzed automatically using configuration management and static analysis tools. Personnel validate the data through a series of steps including training, interviews, independent crosschecks, tool instrumentation, and reviews of results by external researchers. Personnel organize the validated data into a relational database for investigation, analysis, and evaluation by internal NASA personnel as well as external researchers [Sel88] [Wei85].

Using the template, the SEL data sharing arrangement is:

- Lifetime: the data is shared on an ongoing basis with no lifetime restrictions
- Data: project- and module-level data are shared in three primary categories: static code analysis, effort, and faults/changes
- Transfer to third party: transfers are not allowed
- Publication: publications are encouraged and attribution to the SEL is required
- Help: resource limitations constrain the level of support available, but external researchers are encouraged to coordinate and collaborate on results with SEL researchers
- Costs: there is no specific cost structure for sharing
- Derivatives: external researchers are encouraged to coordinate and collaborate on any potential derivatives with SEL researchers

4 Example Data Collection and Sharing Environment: Center for Software Engineering (CSE)

The CSE environment has over 30 industry and government affiliates that help researchers investigate technology, applications, and economic trends through participation in annual workshops and collaborative projects. These interactions cover such areas as value-based spiral model extensions, COCOMO cost-schedule-quality estimation model extensions, software architecture languages and tools, COTS-based system development, and agile methods. In most interactions, industry affiliates contribute project data to the CSE that reflect past and current systems and experiences. The type of data sharing varies based on the research needs and interaction methods, but one common example is the sharing of COCOMO-type parameters and predicted and actual project values [Boe81] [Boe95]. In addition, experience-based Delphi sessions define initial values for new parameters and models. Researchers work together with industrial affiliate project personnel to understand and validate project data at various points throughout development

projects. Researchers organize the validated data into a relational database for investigation, analysis, and evaluation. Among other benefits, the affiliates gain visibility into early and evolving results and models prior to their broader distribution.

Using the template, the CSE data sharing arrangement is:

- Lifetime: the data is shared on an ongoing basis with no lifetime restrictions
- Data: project-level data are typically shared, and there is usually a specific purpose for the data sharing such as the development of a specific model
- Transfer to third party: transfers are not allowed
- Publication: publications of model definitions and results are encouraged and attribution to the affiliates is required
- Help: resource limitations constrain the level of support available, but extensive interactions occur between affiliates and CSE researchers to understand the data and interpretations
- Costs: industry and government organizations pay an annual fee to become a CSE affiliate and collaborate with the researchers
- Derivatives: any potential derivatives are coordinated among the affiliates and researchers to ensure validity of the data and usage

5 Example SEL Empirical Study on Software Reuse

Software reuse enables developers to leverage past accomplishments and facilitates significant improvements in software productivity and quality. Software reuse catalyzes improvements in productivity by avoiding redevelopment and improvements in quality by incorporating components whose reliability has already been established. This example study [Sel05] addresses a pivotal research issue that underlies software reuse – what factors characterize successful software reuse in large-scale systems? The research approach is to investigate, analyze, and evaluate software reuse empirically by mining software repositories from a NASA software development environment that actively reuses software. This software environment successfully follows principles of reuse-based software development in order to achieve an average reuse of 32% per project, which is the average amount of software either reused or modified from previous systems. We examine the repositories for 25 software systems ranging from 3000 to 112,000 source lines from this software environment. We analyze four classes of software modules: modules reused without revision, modules reused with slight revision (< 25% revision), modules reused with major revision (≥ 25% revision), and newly developed modules. We apply non-parametric statistical models to compare numerous development variables across the 2954 software modules in the systems. We identify two categories of factors that characterize successful reuse-based software development of large-scale systems: module design factors and module implementation factors. We also evaluate the fault rates of the reused, modified, and newly developed modules.

The module design factors that characterize module reuse without revision were (after normalization by size in source lines): few calls to other system modules, many calls to utility functions, few input-output parameters, few reads and writes, and many comments. The module implementation factors that characterize module reuse

without revision were small size in source lines and (after normalization by size in source lines): low development effort and many assignment statements. The modules reused without revision had the fewest faults, fewest faults per source line, and lowest fault correction effort. The modules reused with major revision had the highest fault correction effort and highest fault isolation effort as well as the most changes, most changes per source line, and highest change correction effort.

6 Example CSE Empirical Study on Software Cost Modeling

Current software cost estimation models, such as the 1981 Constructive Cost Model (COCOMO) for software cost estimation and its 1987 Ada COCOMO update, have been experiencing increasing difficulties in estimating the costs of software developed to new lifecycle processes and capabilities [Boe81]. These difficulties include non-sequential and rapid-development process models; reuse-driven approaches involving commercial off the shelf (COTS) packages, reengineering, applications composition, and applications generation capabilities; object-oriented approaches supported by distributed middleware; and software process maturity initiatives. This example study [Boe95] defines the baseline COCOMO 2.0 model that supports these new forms of software development. The major new modeling capabilities of COCOMO 2.0 are a tailorable family of software sizing models, involving Object Points, Function Points, and Source Lines of Code; nonlinear models for software reuse and reengineering; an exponent-driver approach for modeling relative software diseconomies of scale; and several additions, deletions, and updates to previous COCOMO effort-multiplier cost drivers. The COCOMO 2.0 model is serving as a framework for an extensive current data collection and analysis effort to further refine and calibrate the model's estimation capabilities.

7 Future Studies Benefiting from Software Data Sharing

An example study underway that is benefiting from software data sharing across projects and organizations is focusing on software requirements metrics [Sel04]. Software requirements metrics are leading indicators of project scope, growth, stability, and progress. Software requirements metrics characterize the "problem space" that a project is addressing, as opposed to metrics such as source-lines-of-code that characterize the "solution space." Software requirements metrics are also available very early in a project and can form the basis for early analyses and predictions of project plans, alternatives, risks, and outcomes. Using software requirements metrics also helps resolve the counting issues associated with reused design or code and whether components are developed in-house or from commercial-off-the-shelf (COTS) suppliers. The project requirements, in terms of functionality and performance, are typically the same regardless of whether the implementation reuses software design and code or incorporates COTS components. Of course, the project requirements, in terms of organization and process, may vary depending on the degree of software reuse and usage of COTS components.

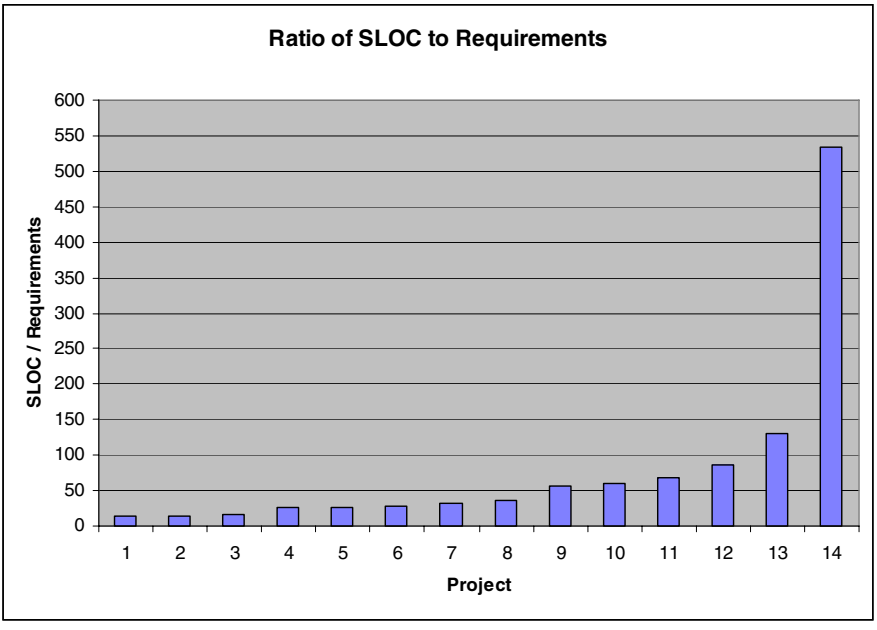


Fig. 1. Software requirements metrics for the projects

The data analyzed originates from 14 large-scale projects that use measurement-driven models to actively manage their development activities and evolving products. Figure 1 displays the software requirements metrics for the projects. For these projects, the number of requirements is defined to be the number of "shall" statements in the requirements specification documents. For example, a requirements document may contain the following statement "the system shall determine the three-dimensional location of a vehicle within an accuracy of 0.1 meter." This "shall" statement would count as one requirement. In order to facilitate consistency within and across projects, requirement specification standards and guidelines need to be defined to enforce the breadth and depth of functionality expressed in a single requirement. Of course, just counting the "shall" statements oversimplifies the project requirements, but this metric does provide an initial basis for project scope, growth, stability, and progress.

Initial data analysis of the software requirements metrics reveals the following observations:

- The ratio of requirements in a system-level parent specification to requirements in a software specification ranges from 1:300 for early projects to 1:6 for mature projects.
- The ratio of requirements in a software specification to delivered source-lines-of-code averages 1:81 for mature projects and has a median of 1:35.
- When Project #14 is excluded (see Figure 1), the ratio of requirements in a software specification to delivered source-lines-of-code averages 1:46 for mature projects and has a median of 1:33.
- Projects that far exceeded the 1:46 requirements-to-code ratio, such as Projects #13 and #14, tended to be more effort-intensive and defect-prone during verification.

8 Conclusions

Our ongoing research investigates principles for effective data collection, analysis, and sharing strategies for enabling software measurement and model building for development and management of large-scale systems. We illustrate a data sharing agreement template and apply it successfully to two data collection environments. The agreement template seems useful for capturing essential elements of data sharing and facilitating fruitful interactions. Successful development, management, and improvement of large-scale systems require the creation of measurement and model building paradigms as well as data collection, analysis, and sharing strategies to support investigations of organizations, projects, processes, products, teams, and resources.

References

[Bas06] V. Basili, M. Zelkowitz, D. Sjoberg, P. Johnson, and T. Cowling, "Protocols in the use of Empirical Software Engineering Artifacts," draft working paper, June 2006.

[Boe81] Barry W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Boe95] B. W. Boehm, B. Clark, E. Horowitz, R. Madachy, R. W. Selby, and C. Westland, "An Overview of the COCOMO 2.0 Software Cost Model," Proceedings of the Seventh Annual Software Technology Conference, Salt Lake City, UT, April 1995.

[Sel88] R. W. Selby and A. A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis," IEEE Transactions on Software Engineering, Vol. SE-14, No. 12, December 1988, pp. 1743-1757.

[Sel04] R. W. Selby, "Software Requirements Metrics Provide Leading Indicators in Measurement-Driven Dashboards for Large-Scale Systems," Proceedings of the 19th International Forum on COCOMO and Software Cost Modeling, Los Angeles, CA, October 26-29, 2004.

[Sel05] R. W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems," IEEE Transactions on Software Engineering, Vol. SE-31, No. 6, June 2005, pp. 495-510.

[Sof83] "Software Engineering Laboratory (SEL): Database Organization and User's Guide, Revision 1," Software Engineering Laboratory, NASA/Goddard Space Flight Center, Greenbelt, MD, technical report no. SEL-81-102, July 1983.

[Wei85] D. M. Weiss and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," IEEE Transactions on Software Engineering, Vol. SE-11, No. 2, February 1985, pp. 157-168.

Knowledge Acquisition in Software Engineering Requires Sharing of Data and Artifacts

Dag I.K. Sjøberg

Abstract. An important goal of empirical software engineering research is to cumulatively build up knowledge on the basis of our empirical studies, for example, in the form of theories and models (conceptual frameworks). Building useful bodies of knowledge will in general require the combined effort by several research groups over time. To achieve this goal, data, testbeds and artifacts should be shared in the community in an efficient way. There are basically two challenges: (1) How do we encourage researchers to use material provided by others? (2) How do we encourage researchers to make material available to others in an appropriate form? Making material accessible to others may require substantial effort by the creator. How should he or she benefit from such an effort, and how should the likelihood of misuse be reduced to a minimum? At the least, the requester should officially request *permission* to use the material, *credit* the original developer with the work involved, and provide *feedback* on the results of use as well as problems with using the material. There are also issues concerning the *protection* of data, *maintenance* of artifacts and *collaboration* among creators and requestors, etc. A template for a data sharing agreement between the creator and requestor that addresses these issues has been proposed.

1 Introduction

A prerequisite for the evolution of most sciences is that researchers build on the work of other researchers. In empirical sciences, this includes the sharing of data and experimental material. For example, to evaluate, compare and generalize results from empirical studies, one should replicate them, and preferably develop theories or models that represent the current knowledge in the field. If replication of studies, meta-analysis, theory development and other research that builds on others' work is stimulated by editors, program chairs and reviewers of journals and conferences, this would be an incitement for individuals to reuse material produced by others.

This chapter is organised as follows. The next section motivates, within the context of empirical software engineering, the need for more replication of studies conducted by other than the original researcher, and the need for more theory building. To support rapid progress in these areas, an increase in the sharing of data and artifacts among software engineering researchers would be required. However, there are several challenges to such a sharing, which is the topic of the subsequent section. Then follow a section that describes a concrete proposal for a template for a data sharing agreement that may help ensure that the reuse of data and material is performed in a way that is satisfactory for both creators and users. The conclusion section ends the chapter.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 77 – 82, 2007.

2 Motivation

The purpose of this section is to demonstrate two important areas in empirical software engineering, replication and theory building, whose progress is dependent on the sharing of data and artifacts among several researchers.

2.1 Replication of Studies

"Methodological authorities generally regard replication, or what is also referred to as "repeating a study," to be a crucial aspect of the scientific method" [1]. In a literature survey, 113 experiments were identified in 103 articles extracted from of a total of 5,453 articles published in major software engineering journals and conferences in the decade 1993-2002 [2]. Only 18 percent of the surveyed experiments were replications. Moreover, of the 20 replications, five can be considered as close replications in the terminology of Lindsay and Ehrenberg [1], i.e., one attempts to retain, as much as is possible, most of the known conditions of the original experiment. The other replications are considered to be differentiated replications, i.e., they involve variations in essential aspects of the experimental conditions. One prominent variation involves conducting the experiment with other kinds of subject (for example, professionals instead of students, undergraduates instead of graduates, etc.), application system, task, etc. Table 1 shows that experiments that are replicated by the same authors tend to confirm the results of the original experiments, and experiments that are replicated by others tend to have different results. This may indicate that when you replicate your own experiments, it is difficult not to be biased. Consequently, replications should preferably be conducted by others.

Table 1. Proportion of differentiated replicated studies that confirm result of the original study

Result	Same authors	Other authors	Total
Confirmation	7	1	8
Different results	1	6	7
Total	**8**	**7**	**15**

2.2 Theory and Model Building

There are many arguments in favour of theory use, such as structuring, conciseness, precision, parsimony, abstraction, generalisation, conceptualisation and communication [3,4,5]. Such arguments have been voiced in the software engineering community as well [6,7,8,9]. Theory provides explanations and understanding in terms of basic concepts and underlying mechanisms, which constitute an important counterpart to knowledge of passing trends and their manifestations. When developing better software engineering technology for long-lived industrial needs, building theory is a means to go beyond the mere observation of phenomena, and to try to understand *why* and *how* these phenomena occur. In general, hypotheses in software engineering are often isolated and have generally a much simpler form than has a theory:

Hypothesis: Technology (process, method, technique, tool language) *A* is better than technology *B*.

Theory: When and why is *A* better than *B,* and how much? Depending on category of developers, tasks, systems, support materials and technology, company culture and other environmental factors, *A* is *X* percent better than *B*, because … etc.

A systematic review of the explicit use of theory in the set of 103 articles reporting controlled experiments (described above), was conducted by Hannay *et al.* [10]. Of the 103 articles, 24 use a total of 40 theories in various ways to explain the cause-effect relationship(s) under investigation. Only two of the extracted theories are used in more than one article, and only one of these is used in articles written by different authors. Hence, there is little sharing of empirically-based theories within the software engineering community, even within topics.

3 Challenges of Sharing Data and Artifacts

We have identified several challenges related to the sharing of artifacts [11]. They concern the *permission* to use the items, the *credit* that should be given to the original creator, the opportunities for *collaboration* between the original creator and the requestor, the kind of *feedback* on the results of use, as well as problems with using the artifact or data that should be reported to the original creator, the *protection* of the data and artifacts, and the *maintenance* of these artifacts:

Permission: Does one have to request permission to use the material? Is it simply publicly available? What should be the rules? If publicly available, how (or should) one provide some form of controlled access to the artifacts? There might be a request to use the artifact with a commitment to provide feedback after or during use (method, results, other data) and reference the items in all work using them. A mechanism that could effectively restrict access would be to require that the requestor write a short proposal to the data owner. Then the item can be used:

- freely, in the public domain,
- with a data sharing agreement or license, or
- with a service fee for use (by industry) to help maintain the data.

Credit: How should the original group gathering the data or developing the artifact be given credit? What would be the rewards for the artifact or data owner? The type of credit is related to the amount of interaction. If there is an interaction, depending on the level, co-authorship may be of value. If it is used without the support of the data owner, some credit should still be given, e.g., acknowledge and reference the data owner. Thus, if the requestor uses it but the owner is not interested in working on the project, the minimal expectation is a reference or an acknowledgement. (There are various possibilities for how that reference should be made, e.g., the paper that first used the artifact, a citation provided by the artifact or data developer, or some independent item where the artifact itself exists a reference.) It is also possible that some form of “associated” co-authorship might be appropriate.

Collaboration: In general, it has been suggested that the requestor keep the option open of collaboration on the work. Funding agencies are often looking for "new" ideas, so it is often difficult to be funded for a continuing operation. What options are there for funding collaborations? If collaboration is not desired by the owner of the artifacts, what are the rights of the requestor? It is probably too strong to require collaboration as a requirement for any requestor.

Feedback: By requiring permission, there is a sense that the originators of the material know that someone is using their materials. However, some form of feedback can act as payment, i.e., updated versions of artifacts, data so it can be used in some form of meta-analysis, some indication of the effectiveness of technology on the experimental environment. A related issue is assuring that the quality of the data, analysis, and new knowledge being returned to the originator is acceptable and consistent within the context of the original experiment.

Protection: There are a large number of issues here. How does one limit potential misuse? How does one support potential aggregation and assure it is a valid aggregation. How does one deal with proprietary data? What about confidentiality? What is required of the originators? Should they be allowed to review results before a paper is submitted for external publication? Does the artifact owner have any rights to stop publication of a paper with invalid results based upon the original artifacts or is the "marketplace of ideas" open to badly written papers? Should there be some form of permission required by reviewers? Who has the rights to analyze and synthesize and create new knowledge based upon the combined results of multiple studies? Again here, how is credit given, authorship determined? How does one limit potential misuse? On the other hand, how do we protect scientific integrity? If users of data find gross negligence on the part of those who created it, what are their obligations to reveal those issues (e.g., the South Korean scandal over stem cell research[1])? Can licensing requirements be an impediment imposed by the guilty to hide their actions?

Maintenance: A large physical device (e.g., particle accelerator) generally is built and supported over the long term. But the same has not been true of computer software, which has an ethereal quality of simple residing hidden in a computer file system. Who pays for the cost of maintaining the experience base? There are only three possibilities here toward maintenance: (1) Owner of the data, (2) Users via a licensing fee, (3) Everyone via an open source arrangement. "Owner of the data" generally will not work since few have such resources, "Licensing fee" may work, but costs will limit use; researchers will not generally pay for something they view as a "free resource." "Open source" is a possibility.

4 A Proposal for a Data and Artifact Sharing Agreement

To help address the challenges described above, a template for a data and artifact sharing agreement has been proposed [11]. The template is shown in Table 2. It has been developed on the basis of experiences from several projects in which data and

[1] http://en.wikipedia.org/wiki/Hwang_Woo-Suk

Table 2. Data/artifact sharing agreement taxonomy

Attribute	Property	Value	Definition
Lifetime	Permission	Single use	Can use artifact only for one application
		Limited	Can use artifact repeatedly for a set period of time
		Unlimited	Unlimited use of the artifact
Area	Permission	Specific project	Can use artifact only for one project
		Specific research	Can use artifact within one research area
		Unlimited	Unlimited use of the artifact
Data	Protection	Sanitized	No personal information contained
		Proprietary	Data contains information that uniquely identifies individuals of specific organizations
Transfer to 3rd party	Permission	No	Only signer of agreement can use artifact
		Yes	Signer of agreement can pass on artifact under the same agreement conditions to another. This may require a non-disclosure agreement with either this signer or owner of artifact.
		Yes after period	Signer of agreement can pass on artifact after a period of time (e.g., restricted for 3 years then available to anyone)
Publication	Credit, Feedback	None	Signer of agreement is free to use artifact in any way.
		Prior results	Signer of agreement has to send results of using artifact to owner of artifact prior to writing a paper on the topic
		Acknow-ledge	Signer of agreement has to acknowledge creator of artifact in publication. Agreement will state how this acknowledgement will occur.
		Review	Artifact owner has rights to review paper based on artifact prior to publication submission
Help	Collabo-ration	Data only	Signer of agreement obtains the data "as is." No help is provided from artifact owner.
		Limited	Artifact owner is willing to provide limited help to signer of agreement to use artifact.
		Extensive	Artifact owner is willing to provide significant collaboration and may want to be co-author on publications.
Costs	Maintenance	None	Artifact is free to signer of agreement, with perhaps a minimal cost for a tape or CD of data
		Payment	A set amount is specified to obtain artifact. If successful, this may help provide funding for maintenance of artifact repository.
Derivatives	Permission, Feedback, Protection, Maintenance	None	Derived artifact is owned by signer of agreement. (May be separate clauses covering derived software and related artifacts or derived data using meta-analysis)
		Creator	Derived artifact is owned by original artifact creator and creator must get a copy of the derived artifact.
		Open-source	An agreement such as used by the open source community from the Free Software Foundation. Any derived work has the same usage requirements as the original artifact.

artifact sharing has been undertaken [11]. I have used it successfully myself in two recent projects: one small project where most of the attributes were not considered relevant, and another, larger project in which we had to include many details that

were not in the template. Note that the purpose of the template is to provide a general framework that in most cases would need adaptations depending on the actual project.

5 Conclusions

To make progress in empirical software engineering, we need to build on the work of each other; we need to share data, testbeds and other artifacts. The proposed basic data sharing agreement must evolve based on the feedback from actual use. Hence, we hope that as many as possible will start using this template and report experiences and suggestions for change to the author of this chapter or one of the authors of [11].

References

1. R.M. Lindsay and A.S.C. Ehrenberg, The Design of Replicated Studies, The American Statistician, vol. 47, pp. 217-228, Aug. 1993.
2. D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanović, N.-K. Liborg, and A.C. Rekdal. A survey of controlled experiments in software engineering. IEEE Transaction on Software Engineering, 31:733–753, September 2005.
3. S.B. Bacharach. Organizational theories: Some criteria for evaluation. Academy of Management Review, 14(4):496–515, 1989.
4. J.W. Lucas. Theory-testing, generalization, and the problem of external validity. Sociological Theory, 21:236–253, 2003.
5. D.G. Wagner. The growth of theories. In M. Foschi and E.J. Lawler, editors, Group Processes, pages 25–42. Nelson–Hall Publishers, Chicago, 1994.
6. V.R. Basili. Editorial. Empirical Software Engineering, 1(2), 1996.
7. A. Endres and D. Rombach. A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories. Fraunhofer IESE Series on Software Engineering. Pearson Education Limited, 2003.
8. B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. IEEE Transaction on Software Engineering, 28(8):721–734, August 2002.
9. W.F. Tichy. Should computer scientist experiment more? 16 excuses to avoid experimentation. IEEE Computer, 31(5):32–40, May 1998.
10. J.E. Hannay, D.I.K. Sjøberg, T. Dybå, A Systematic Review of Theory Use in Software Engineering Experiments, IEEE Transaction on Software Engineering, 33(2): 87–107 February 2007.
11. V. Basili, M. Zelkowitz, D. I.K. Sjøberg, P. Johnson and T. Cowling, Protocols in the use of Empirical Software Engineering Artifacts, Empirical Software Engineering. 2007. (forthcoming)

Effective Data Interpretation

Jürgen Münch

Abstract. Data interpretation is an essential element of mature software project management and empirical software engineering. As far as project management is concerned, data interpretation can support the assessment of the current project status and the achievement of project goals and requirements. As far as empirical studies are concerned, data interpretation can help to draw conclusions from collected data, support decision making, and contribute to better process, product, and quality models. With the increasing availability and usage of data from projects and empirical studies, effective data interpretation is gaining more importance. Essential tasks such as the data-based identification of project risks, the drawing of valid and usable conclusions from individual empirical studies, or the combination of evidence from multiple studies require sound and effective data interpretation mechanisms. This article sketches the progress made in the last years with respect to data interpretation and states needs and challenges for advanced data interpretation. In addition, selected examples for innovative data interpretation mechanisms are discussed.

1 Introduction

Software practitioners and researchers increasingly face the challenging task of effectively interpreting data for project control and decision making, and gaining knowledge on the effects of software engineering technologies in different environments. This is caused, for instance, by the increasing necessity to use quantitative approaches in practice in order to climb up maturity ladders or the need for justifying software-related costs in the context of business strategies and business value. In the area of empirical research, there is a need to come up with sufficiently general, yet significant context-oriented evidence on the effects of software technologies based on data from individual or multiple studies.

This article focuses on three areas where data interpretation is relevant: (1) Interpretation of data for project control. Here, the focus is on project execution. Factors such as the increasing distribution of development activities, the need for monitoring risks, or regulatory constraints have accelerated the introduction of data-based project management techniques into practice. However, making valuable use of collected data is challenging and requires effective mechanisms for data interpretation. (2) Interpretation of data for individual empirical studies. Due to the specifics of software engineering studies, the data gained from such studies typically does not allow for the application of statistical analysis and interpretation techniques that are successfully applied in other fields (e.g., methods that require a significant amount of normally distributed data). Methods for data analysis and interpretation are needed that can cope with typical specifics of software engineering data. (3) Combination of evidence. Here, the focus is on aggregating evidence from multiple individual studies. Data or

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 83 – 90, 2007.

results from different individual studies are typically heterogeneous and stem from different contexts. Interpreting data in order to gain aggregated combined evidence requires strategies and techniques to cope with these difficulties.

For these three areas, the article sketches the progress made in the last years with respect to data interpretation and states needs and challenges for advanced data interpretation. In addition, selected examples for innovative data interpretation mechanisms are discussed.

2 Data Interpretation for Project Control

Measurement is an important means for managing software development and maintenance projects in a predictable and controllable way. This requires particularly accurate and precise monitoring of process and product attributes. Systematic support for detecting and reacting on critical project states in order to achieve planned goals is needed. Single points of control are required to monitor, coordinate, and synchronize distributed development activities.

Progress

During the last years, we have observed several trends that are relevant for data interpretation for project control, especially:

- Increased industry awareness for data-driven project management and quantitive approaches. This is partially motivated by the application of maturity models, but there are also other reasons such as distributed development and globalization.
- Dashboards are currently being widely installed in industry. One of the reasons is that regulatory constraints often require higher process transparency.
- Measurement has begun to enter the acquisition process. There is, for instance, a trend for OEMs in the automotive industry to enforce measurement-based assessment of supplier software.
- Software is increasingly entering domains (such as transportation systems or medical engineering) that demand quantitative assurance of critical processes and product properties.

Selected Needs

We see the following selected needs as being important with respect to data interpretation for project control:

- Establish quantitative project control mechanisms.
- Obtain single point of control.
- Define process interfaces quantitatively.
- Integrate business and engineering processes. Currently, project controlling on the engineering level and on the higher management level are widely separated. Linking business goals to goals of the software organization of a company and to measurement goals is necessary for integrating these two levels of control.

Challenges

We see the following essential research challenges with respect to data interpretation for project control:

- How to interpret data in the context of software goals and business goals?
- How to visualize data in a purpose-, role-, and context-oriented manner?
- How to tailor and combine analysis, interpretation, and visualization techniques?
- How to integrate heterogeneous data from different sources?

Example: Software Project Control Centers (SPCC)

One means to institutionalize measurement on the basis of explicit models (i.e., process models, product models, resource models, and quality models) is the development and establishment of so-called software project control centers (SPCC) for systematic quality assurance and management support [7,9,10]. An SPCC can be defined as a means for process-accompanying interpretation and visualization of measurement data: It consists of (1) underlying techniques and methods to control software development projects and additional rules to select and combine them, (2) a logical architecture that defines logical interfaces to its environment, and (3) supporting tool(s) that implements (parts of) the logical architecture. Its input information includes, but is not limited to, information about project goals and characteristics, project plan information (e.g., target values per development phase), measurement data of the current project, and empirical data from previous projects. Its output information includes a context-, purpose-, and role-oriented visualization of collected and interpreted measurement data. That is, the visualization depends upon the context of the project, the purpose of the usage (e.g., monitoring), and the role of the user project manager. Its tasks include collecting, interpreting, and visualizing measurement data in order to provide context-, purpose-, and role-oriented information for all involved stakeholders (e.g., project managers, quality assurer, developers) during the execution of a software development project. This includes, for instance, monitoring profiles, detecting abnormal effort deviations, cost estimation, and cause analysis of plan deviations.

3 Data Interpretation for Individual Studies

Understanding the effects of software engineering techniques and processes under varying conditions can be seen as a major prerequisite towards predictable project planning and guaranteeing software (or system) quality. Evidence regarding the effects of techniques and processes for specific contexts can be gained by individual studies. Due to the fact that software development is a human-based and non-deterministic activity, the data gained in such studies typically has several limitations (e.g., limited validity and completeness) and is context-dependent. Effective data interpretation has to cope with this and support the derivation of results that are sufficiently general on an acceptable significance level.

Progress

During the last years, we have observed several trends that are relevant for data interpretation for individual studies, especially:

- New or enhanced analysis techniques and tools, e.g.,
  - for analysis of little and/or imperfect data sets
  - for combining quantitative data and expert opinion
  - for data mining
- Tools with new or enhanced capabilities, e.g.,
  - visualization tools (isolated cases)
  - tailorable product (and process) measurement tools

Selected Needs

We see the following selected needs as being important with respect to data interpretation for individual studies:

- Effectively interpret results for different stakeholders.
- Effectively develop or calibrate quantitative models for different purposes (e.g., reliability prediction).
- Preprocess imperfect data sets appropriately as prerequisite for applying data analysis techniques.

Challenges

We see the following essential research challenges with respect to data interpretation for individual studies:

- How to combine different analysis/interpretation techniques (e.g., statistical analysis and visualization)?
- How to guarantee data validity in industrial settings?
- How to preprocess imperfect data sets for analysis and interpretation?
- How to select quantitative models based on goals and available data?
  - What kind of data is needed?
  - Gap analysis: What data is missing for building the models?
- How to visualize data appropriately?
  - What are appropriate metaphors? Can they be standardized to a certain extent?

Example: Visualization

One approach to data interpretation is to use the visual capabilities of people. Visualization mechanisms support the understanding of the data and aspects under consideration, the abstract and compact representation of information, and the creation of a mental model of the data. Special visual environments have interactive capabilities and allow, for instance, easy navigation through the data by flexibly changing perspectives and abstraction levels (see, for instance, [14]).

4 Combination of Evidence from Individual Studies

One of the challenges of empirical research is to overcome the typically narrow scope of validity of the results. From the viewpoint of a practitioner, an important question is whether the results are valid for his own development context. A promising way to broaden the scope of empirical evidence is to summarize and organize evidence through integration and aggregation [5]. Integration means accumulating different kinds of evidence — ranging from quantitative results to qualitative practical experiences and human judgements. Aggregation means accumulating evidence from different contexts. Both, integration and aggregation, require effective interpretation mechanisms.

Gaining more evidence about processes, products or qualities should be packaged in explicit models. This requires a process for systematically evolving such models and creating variations of the models, if necessary. Fig. 1 illustrates such a process for evolving models. The process has been proposed by Rombach [6] and can be seen as a basis for packaging models in an experience base [3].

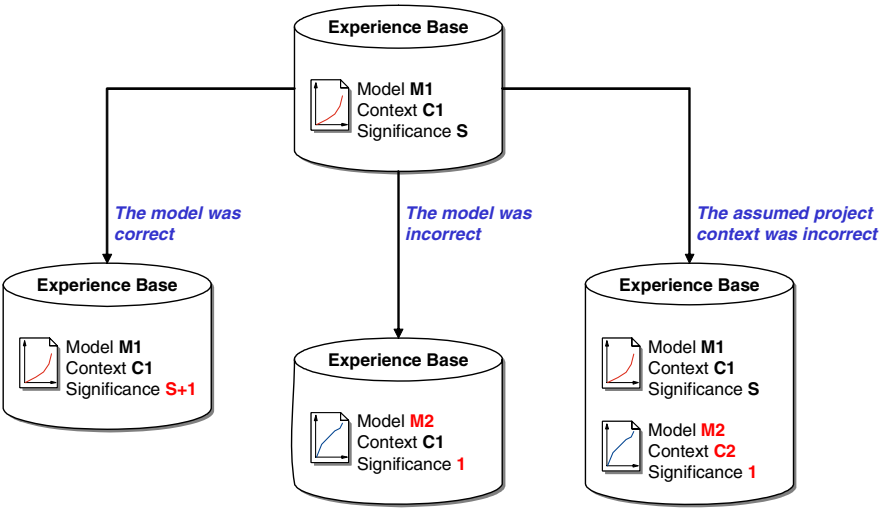


Fig. 1. A Process for Evidence-based Model Evolution [6]

Progress

During the last years, we have observed several trends that are relevant for combining evidence:

- Collections of empirical evidence (handbook, repositories) have been created.
- Many (company-specific) data repositories are available.
- The concept of virtual laboratories was developed.
- Variability concepts for products have been established.

Selected Needs

We see the following selected needs as being important with respect to combining evidence:

- Effectively derive, maintain, and present aggregated trustable evidence and statements (bottom-up).
- Verify aggregated evidence and identify lacks of evidence (top-down).
- Select and customize processes, techniques, tools, and products based on evidence.

Challenges

We see the following essential research challenges with respect to combining evidence:

- How to define appropriate operators for aggregating empirical evidence by taking the project context into account?
- How to present/visualize combined evidence for different stakeholders?
- How to identify lacks of evidence?
- How to reengineer GQM plans [2,13] from data repositories?
- How to describe process variability?
- How to represent available evidence and lacks of evidence for specific context variations?
- How to integrate evidence into process, product, and quality models as well as into tools? Evidence-based decision models for product lines and variant-rich processes are needed.
- How to evolve process, product, and quality models?

Example: Virtual Laboratory

Combining process simulation [1] and real experiments is a promising way to fill the areas of missing evidence between individual studies (e.g., combinations of impact factors that are not covered by real studies). This is addressed by the concept of Virtual Software Engineering Laboratories (VSEL), which was introduced at first in [12] and refined in [11]. One major motivation for such a virtual software engineering laboratory is cost reduction by simulating human behavior and the process environment of the method to be examined. Additionally, such a laboratory allows for better demonstrating the effects of a method in an understandable way. In particular, a multitude of variations of an experiment that is often necessary to cover different impact factors can be performed, and costs can be reduced enormously. Consequently, learning cycles can be shortened. In particular, empirical studies and process simulation can be combined in such a way that 1) empirical knowledge is used for the development and calibration of simulation models, 2) results from process simulation are used for supporting real experiments, 3) real experiments and process simulation are performed in parallel (i.e., online simulation).

5 Conclusions

Effective data interpretation plays an important role in software project management and for gaining evidence from empirical studies. Appropriate data interpretation, presentation, and dissemination of results can be seen as a major acceptance and success factor for quantitative project management and empirical studies. Concluding, we recommend the following when considering effective data interpretation:

- Make sure that the study is relevant and important before conducting the study ("test first"). For industry, the following questions might be relevant: Is there a need for the evidence? By whom? How will it be used? How does it relate to business goals? What is the cost/benefit relation of gaining the evidence? Is there a dissemination and exploitation strategy? For research, the following questions might be relevant: Is there a lack of evidence? How could the results be combined with other evidence?
- The combination of different analysis and interpretation techniques promises to broaden the scope of the evidence and provide new insights. Example techniques are visualization, simulation, qualitative analysis, quantitative analysis, and meta analysis.
- Consider data interpretation mechanisms early on during the establishment of project controlling mechanisms or the design of empirical studies.

Acknowledgements

I would like to thank Sonnhild Namingha from Fraunhofer IESE for preparing the English editing of this paper. This work was supported in part by the German Federal Ministry of Education and Research (Soft-Pit Project, No. 01ISE07A).

References

1. Armbrust, O., Berlage, T., Hanne, T., Lang, P., Münch, J., Neu, H., Nickel, S., Rus, I., Sarishvili, A., Stockum, S.v., Wirsen, A., "Simulation-based Software Process Modeling and Evaluation", In: Handbook of Software Engineering and Knowledge Engineering, Vol. 3: Recent Advances", (S. K. Chang, ed.), World Scientific Publishing Company, pp. 333-364, August 2005.
2. Basili, V.R., Weiss, D.M., "A Methodology for Collecting Valid Software Engineering Data". IEEE Transactions on Software Engineering 10(6), 728-738, 1984.
3. Basili, V.R., Caldiera, G., Rombach, D., "Experience Factory"; in: Marciniak J.J. (ed.), Encyclopedia of Software Engineering, Vol. 1, John Wiley & Sons, 2001, pp. 511-519.
4. Briand, L.C., Differding, C., Rombach, D., 1996. "Practical Guidelines for Measurement-Based Process Improvement". Software Process: Improvement and Practice 2(4), 253-280.
5. Ciolkowski, M., Münch, J., "Accumulation and Presentation of Empirical Evidence: Problems and Challenges", Proceedings of the 2005 workshop on Realising evidence-based software engineering (REBSE 2005), St. Louis, Missouri, pp. 1-3, USA, May 17, 2005.

6. Heidrich, J., Münch, J., Riddle, W.E., Rombach, D., “People-oriented Capture, Display, and Use of Process Information”, In: New Trends in Software Process Modeling, (S. T. Acuña and M. I. Sánchez-Segura, eds.), Series on Software Engineering and Knowledge Engineering, Vol. 18, World Scientific Publishing Company, pp. 121-179, 2006.
7. Heidrich, J., Münch, J., Wickenkamp, A., “Usage-Scenarios for Measurement-based Project Control”, Proceedings of the 3rd Software Measurement European Forum (SMEF 2006), (Ton Dekkers, Ed.), pp. 47-60,Rome, Italy, May 10-12, 2006.
8. Kellner, M.I., Madachy, R.J., Raffo, D.M. 1999. “Software process simulation modeling: why? what? how?”, Journal of Systems and Software 46(2/3): 91–105.
9. Münch, J., Heidrich, J., “Software Project Control Centers: Concepts and Approaches”, International Journal of Systems and Software, vol. 70, issues 1-2, pp. 3-19, February 2004.
10. Münch, J., Heidrich, J., “Tool-based Software Project Controlling“, In: Handbook of Software Engineering and Knowledge Engineering, Vol. 3: Recent Advances”, (S. K. Chang, ed.), World Scientific Publishing Company, pp. 477-512, August 2005.
11. Münch, J., Pfahl, D., Rus, I., ”Virtual Software Engineering Laboratories in Support of Trade-off Analyses”, International Software Quality Journal, Special Issue on “Trade-off Analysis of Software Quality Attributes”, vol. 13, no. 4, pp. 407-428, December 2005.
12. Münch, J., Rombach, D., Rus, I., “Creating an Advanced Software Engineering Laboratory by Combining Empirical Studies with Process Simulation”, Proceedings of the 4th International Workshop on Software Process Simulation and Modeling (ProSim 2003), Portland, Oregon, USA, May 3-4, 2003.
13. Rombach, D., 1991. “Practical benefits of goal-oriented measurement”. Software Reliability and Metrics, 217-235.
14. Schäfer, T., Mezini, M., “Towards More Flexibility in Software Visualization Tools”, in Proc. VISSOFT’05, IEEE CS Press, 2005.

Software Support Tools and Experimental Work

Audris Mockus

1 Introduction

Presently it is difficult to imagine a software project without version control and problem tracking systems. This may be partly attributed to the emergence of open source projects where participants never meet each other and increasingly popular commercial globally distributed projects where groups of developers are often separated by many time zones. Version control and problem tracking tools are a basic necessity in such communication-poor environments. However, their value is getting more and more recognized in small co-located projects because information stored in these tools represent most of the project's decision making history.

Therefore, the idea to use repositories of software support tools to explain and predict phenomena in software projects and to create tools that improve software productivity, quality, and lead times appears to be promising. This is particularly salient to many open source software projects where all project related discussion and decisions are externalized in the mailing lists and other tools. In many such projects it is considered inappropriate to discuss project matters in private discussions not recorded on the relevant mailing lists.

The study of open source projects has other significant motivations as well. Such projects present a conundrum from software engineering perspective as they apparently lack key aspects such as requirements and design that are thought to be essential for project's success. The motivation of volunteer participants can not be convincingly explained using current economic theories. Open source is considered to be the new technological commons that, instead of being destroyed by over-use, is, on the contrary, benefiting from it. Tragedy of the commons is a concept of individually optimal decisions (for example, having a lager herd) leading to suboptimal outcomes for everyone (common grazing lands destroyed).

A typical analysis of software repositories includes retrieval, summarization, and validation of data from software project's version control and problem tracking systems. Unfortunately, extracting, cleaning and validating, and drawing conclusions from such data poses formidable challenges because data sources are not designed as measurement tools, and the tools involved as well as practices of using the tools vary from project to project.

The topic has recently attracted substantial attention including a special issue of the Transactions on Software Engineering (Vol. 31, No. 6) and an annual workshop on "Mining Software Repositories".

Here we attempt to outline the overall methodology and list some of the opportunities and challenges of using project support systems in empirical work. We start from the overview of the tools used, continue with methodology and its benefits, and conclude with the list of remaining challenges.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 91 – 99, 2007.

2 Tools Supporting Software Development

Although software tools are used to support virtually any software development project there are important differences in the tools and the ways they are used. The first broad distinction can be made between open source projects and commercial projects. Open source projects tend to use three core tools. Version control systems such as CVS [3] (and now, increasingly, Subversion [4]) are used to keep track of the changes to the code and to grant permission to make changes to project members. Every change is usually accompanied by an automatic email sent to the special change mailing list to notify other project participants. Problem tracking systems, such as Bugzilla or Scarab, provide a way to report and track the resolution of issues. Finally, mailing lists provide a forum to discuss issues other than changes or problems. Most projects have a developer mailing list to discuss features, design, and implementation, and a user mailing list to discuss installation and usage of the product.

Commercial projects tend to contain more numerous and varied tools to track various aspects of the development and deployment processes. Although some of these systems contain little information helpful in analyzing software production this may change as the objectives and scope of the analyses evolve in the future. Sales and marketing support systems may contain customer information and ratings, purchase patters, and customer needs in terms of features and quality. The accounting systems that track purchases of equipment and services contain information about installation dates for releases. Maintenance support systems should have an inventory of installed systems and their support levels. Field support systems include information about customer reported problems and their resolution. Development field support systems contain software related customer problem tracking and patch installation tracking. Development support systems are similar to open source projects and contain feature, development, and test tracking. Common version control tools in commercial environments include ClearCase [25] and Source Code Control System (SCCS) [26] and its descendants. Most projects employ a change request management system that keeps track of individual requests for changes, which we call Modification Requests (MRs). Problem tracking systems, unlike change management systems, tend not to have built-in relationship between MRs (representing problems) and changes to the code. Extended Change Management System (ECMS) [14] is an example of change management system that uses SCCS for version control.

Large software products employ a number of service support tools to help predicting and resolving customer problems. Such systems may be used to model software availability [17], though their description is beyond the scope of this presentation.

3 Basic Methodology

The amount and complexity of available data necessitates the use of analysis tools except, possibly, in the smallest projects. Such analysis systems contain the following capabilities [8]:

- Retrieve the raw data from the underlying systems via access to the database used in the project support tools or by "scraping" relevant information from the web interfaces of these systems. For example, CVS changes can be obtained via cvs log command, and Bugzilla data is stored in MySQL relational database.
- Clean and process raw data to remove artifacts introduced by underlying systems. Verify completeness and validity of extracted attributes by cross-matching information obtained from separate systems. For example, match changes from CVS mail archives, from cvs log command or matching CVS changes to bug reports and identities of contributors.
- Construct meaningful measures that can be used to assess and model various aspects of software projects.
- Analyze data and present results.

These capabilities represent processing levels that help users cope with complexity and evolution of the underlying support systems or the evolution of the analysis goals by separating these concerns into separate levels that can (and should) be validated independently. Each level refines data from the previous stage producing successively better quality data, however it is essential that links to raw data are retained to allow automatic and manual validation.
In summary, the key desirable features of the analysis include:

1. Iterative refinement of data with each iteration obtaining quantities that may be more interpretable and more comparable across projects.
2. Each item produced at every stage retains reference to raw data to facilitate validation.
3. Each processing level has tools to accomplish the step and validation techniques to ensure relevant results. Because the projects and the processes may differ, it is essential to perform some validation on each new project.

The main stages and the tools needed to perform them are described in the subsections below. More detail on tools used for open source projects may be found in [15].

3.1 Development Process

The changes to the source code tend to follow a well-defined process. Unfortunately, that process may greatly vary with project, therefore it has to be obtained from project development FAQ or from another document on development practices. The practices need to be validated by interviewing several developers and testers (or other participants administering or using project support tools) on a small subset of their recent MRs or changes.

In rough terms, the new software releases or software updates are product deliveries that contain new functionality (features) and fixes or improvements to the code. Features are the fundamental design unit by which the system is extended. Large changes that implement a feature or solve a problem may be decomposed into smaller self-contained pieces of work often called modification requests (MRs).

To perform the changes a developer or a tester (or, in open source projects even the end user) can "open" MRs. The MR is then assigned (or self-assigned) to a developer who makes the modifications to the code (if needed), checks whether the changes are

satisfactory, and then submits the MR. In some instances of development processes the tester or code owner may then inspect and approve the MR. The MR ID is usually included in a the comments or as a separate field in the attributes of the change recorded by the version control system.

Version control systems (VCS) operate over a set of source code files. An editing change to an individual file is embodied in a delta: conceptually the file is "checked out," edited and then "checked in." An atomic change, or delta, to the program text consists of the lines that were deleted and those that were added in order to make the change. Deltas are usually computed by a file differencing algorithm (such as Unix diff), invoked by the VCS, which compares an older version of a file with the current version. Included with every delta is information such as the time the change was made, the person making the change, and a short comment describing the change. Whereas a delta is intended to keep track of lines of code that are changed, an MR is intended to be a change made for a single purpose. Each MR may have many deltas associated with it.

3.2 Retrieval of Raw Data

Software changes are obtained from version control systems and contain developer login, timestamp of the commit, change comment, file, and version id.

Once the set of revisions (and their attributes) are extracted, the underlying code changes can be extracted by obtaining all versions of all files and the differences between their subsequent versions. This is the most involved operation but it allows fully to reconstruct the code evolution in each file and is necessary to obtain the exact content of each change. The content of a change can then be used to determine if the change involved comments or code and to identify what functions or statements were changed.

Problem (or bug) reports typically contain MR ID, severity, development stage at which the problem was detected, software release, description, status history (identity of individuals, dates, and status changes), and various attachments needed to explain the nature of the problem or the way it was resolved. Unlike CVS, most problem tracking systems store information in a relational database. If access to such database is difficult to obtain, it may be possible to retrieve web pages for each problem report and extract the relevant attributes from these web pages.

Mailing lists tend to be archived and are, therefore, easy to download and process. Tools that identify and count threads, extract relevant dates, and patches may be helpful.

Extracting raw data, although time consuming, contains few pitfalls. However, different projects may use slightly different format or slightly different attributes even if they use the same systems. The most common issue is likely to be that network congestion or version control locking issues may prevent obtaining the full set of items. A more robust option is to retrieve data in smaller chunks.

3.3 Augmenting Raw Data

System generated artifacts involve data points that do not represent activity of interest. An example of such artifact is an empty delta where the code is not modified. Such changes are common when creating branches in the version tree. Another

common problem is copying of CVS repository files. In such case, duplicate delta from two or more files that had the same origin or are included in several modules are eliminated.

MRs are groups of delta done to perform a particular task. In cases where MR is indicated in the comment of a delta such grouping can be established. Many projects using CVS do not follow that practice. A common approach is to group delta that share login and comment and are submitted within a brief time period. The drawback of this approach is difficulty of determining the right interval to create breaks and the possibility that delta with different comments may belong to the same MR. A sample of groupings needs to be inspected to determine the most suitable time window.

In cases where MR IDs (or other information needed for the analysis) are embedded within delta comment their ID has to be extracted from the comment and associated with appropriate delta. This tends to be fairly straightforward using pattern matching techniques because MR IDs have a well defined format.

The step of augmenting raw data may involve cleaning attribute values that are entered manually, because manual input is always associated with errors. If, for example, a release number is typed (instead of selected from a list of choices), it may be necessary to inspect all unique values (in their frequency order) and process the output to change at least the most frequent misspellings to their intended values.

3.4 Producing Change Measures

Before various measures are produced, the semantics of the attributes may need clarification. For example, each MR may be associated with a release where the problem was found and with all releases where it was fixed. Many large projects track the problem not just for a release where it was found but also for the future (and in rare cases for the past) releases where the problem may manifest itself. Typically, the problem is first resolved for the release where it has been reported because subsequent releases are often still in development stages. This has important implications for measurement. If we are analyzing the number of problems found in a release, we have to count such MR only once for the release it was detected in. However, if we are looking at effort and schedule, we have to investigate all MRs resolved in a particular release because it requires effort to resolve the same MR for each release. Therefore, MRs resolved for several releases would affect effort and schedule for each release.

Several change measures are described in [21]. Change diffusion measures the files, modules, and subsystems affected by the change or the developers and organizations involved in making the change. Change size may be operationalized as the number of lines of code (LOC) added or deleted and LOC in the files touched by the change. A convenient proxy for both size and diffusion is the number of delta in a change. The duration of a change may be measured conservatively as the time elapsed between the first and the last delta or by comparing MR creation and submission or resolution dates. Often it is helpful to separate bug fix MRs from MRs used to track new features [20] and identify MRs associated with customer reported problems. Measures of experience (number of delta) or productivity (number of MRs resolved per year) can be associated with a developer or organization and measures of faultiness (fraction of MRs reported by customers or post unit-test) can be associated with files or modules.

The choice of measures may be dictated by the needs of a particular study, but basic summaries tend to be useful in most studies.

3.5 Models and Tools

There has been a substantial amount of work involved in modeling software changes. It was found that past changes are the best predictor of module faults [9] and that the diffusion and expertise of developers affects the likelihood that a patch will break [21].

The idea that files frequently touched together but not with other files define chunks that can be maintained independently was investigated in the context of globally distributed software development in [22] and it was found that MRs spanning such chunks take longer to complete [11]. It turns out that MRs involving geographically separated developers take more than twice as long to complete [12].

Models of expertise and relevance can provide most relevant experts [19], most relevant files [27, 13], or most relevant defects [5].

A general topic of evaluating the effect of software engineering tools and practices relies on the ability to estimate developer effort [10]. Work in [1, 2, 7] investigates the effort savings of version sensitive editor, visual programming environment, and refactoring of a legacy domain.

A number of hypotheses on how open source and commercial software engineering practices differ and their effect on productivity quality and lead time are investigated in [18, 6].

At a much higher level entire releases are modeled via changes to predict release schedule [23]. The work assumes that a random number of fix MRs are generated with a random delay from each new feature MR. An assumption that the work flow of MRs in the past releases is similar to the work flow of the current release is used to predict release readiness [16].

A probability that a customer will observe a failure related to a software problem is modeled in [24].

4 Advantages and Pitfalls of Using Project Support Systems

Probably the most obvious advantage of using project support systems such as customer problem tracking system is that the data collection is non-intrusive because such systems are already deployed and used. However, that does not reduce the need for in-depth understanding of a project's development process and, in particular, of how the support systems are used.

A long history of past projects whose data has been captured in project support systems enables historic comparisons, calibration, and immediate diagnosis in emergency situations.

The information obtained from the support systems is often fine grained, at the trouble ticket/software alarm/customer installation level. However, links to aggregate attributes, such as features and releases, is often tenuous.

The information tends to be complete, as every action involving development or support is recorded. However the information about what the action pertains to may be nontrivial to infer and some of the data entries, especially those not essential for the domain of activity, tend to be inconsistently or rarely supplied.

The data are uniform over time as the project support systems are rarely changed because they tend to be business-critical and, therefore, difficult to change without major disruptions. That does not, however, imply that the process was constant over the entire period one may need to analyze.

Even fairly small projects contain large volumes of information in the project support systems making it possible to detect even small effects statistically. This, however, depends on the extractability of the relevant quantities.

The systems are used as a standard part of the project, so the software project is unaffected by experimenter intrusion. We should note that this is no longer true when such data are used widely in organizational measurement. Organizational measurement initiatives may impose data collection requirements that the development organizations might not otherwise use and modify their behavior in order to manage the measures tracked by these initiatives.

The largest single obstacle for using the project systems for analysis is the necessity to understand the underlying practices and the way the support systems are used. This requires validation of the values in fields used by the developers and support technicians to assess the quality and usability of the attribute. Common and serious issues involve missing and, especially, default values that may render an attribute unusable. Any fields that do not have a direct role in the activities performed using the project system are highly suspect and, often provide little value in the analysis. As the systems tend to be highly focused to track issues or versions, extracting reliable data needed for analysis may pose a challenge.

It is worth noting that analyzing data from software support systems is labor intensive. At least 95% of effort should be expected to involve understanding the practices of tool use, cleaning and validating data, and designing relevant measures. There is no guarantee at the outset of the study that the phenomena of interest would be extractable with sufficient accuracy. All too often, such obstacles lead to temptation to model easily-to-obtain yet irrelevant measures, to study phenomena of no practical significance, and to get fascinated by oddities of the tool generated artifacts.

5 Challenges

The motivation to deploy support system based measurement can come from immediate and relatively straightforward applications in project management, such as dashboards showing MR inflow and outflow that help visualize when the project is getting late or to anticipate the completion date.

Although the information in software support systems represents a vast amount of untapped resources for empirical work, it remains a challenge to create models of key problems in software engineering based on that data and to simplify and streamline the data extraction and validation process. This raises a question of how best to improve version control and problem tracking tools to facilitate measurement and analysis. Unfortunately, it is not simply a question of what attributes to add - many fields in the existing systems are empty or contain noise in cases where they provide no clear value to the system users. Therefore, it is of interest to study what information developers would willingly, easily, and accurately enter in problem tracking and version control systems.

It remains to be seen how best to characterize a single software project based on its software repositories and what validation is necessary for that characterization. What models would be plausible for a single software project?

To confirm findings based on studies of an individual project it may be necessary to investigate a larger collection of similar projects. How to minimize the effort needed to validate the compatibility of practices in such a large sample of software projects?

This leads to questions about the role of software repositories in design, planning, execution, and analysis of experiments. Because the models of projects, people, and code tend to be based on the properties of changes it is of interest to know which properties are the most important. In other words, what is the "sufficient statistic" for a software change? A sufficient statistics is a statistical term meaning a summary information that is needed to know all the relevant properties of a sample. For example, a sample from Gaussian distribution can be summarized with just two numeric values - sample mean and sample variance.

Even though it appears that the use of software repositories should enable answering novel software engineering questions, most of these questions have yet to be identified.

References

[1] D.Atkins, T.Ball, T.Graves, and A.Mockus. Using version control data to evaluate the impact of software tools: A case study of the version editor. IEEE Transactions on Software Engineering, 28(7):625-637, July 2002.

[2] D.L. Atkins, A.Mockus, and H.P. Siy. Value Based Software Engineering, chapter Quantifying the Value of New Technologies for Software Development, pages 327-344. Springer Verlag Berlin Heidelberg, 2006.

[3] Per Cedeqvist and etal. CVS Manual. May be found on: http://www.cvshome.org/CVS/.

[4] Ben Collins-Sussman, BrianW. Fitzpatrick, and C.Michael Pilato. Subversion Manual. May be found on: http://svnbook.red-bean.com/.

[5] D.Cubranic and G.C Murphy. Hipikat: A project memory for software development. TSE, 31(6), 2005.

[6] TDinh-Trong and Bieman J.M. Open source software development: A case study of freebsd. IEEE Transactions of Software Engineering, 31(6), 2005.

[7] Birgit Geppert, Audris Mockus, and Frank R□ler. Refactoring for changeability: A way to go? In Metrics 2005: 11th International Symposium on Software Metrics, Como, September 2005. IEEE CS Press.

[8] Daniel German and Audris Mockus. Automating the measurement of open source projects. In ICSE '03 Workshop on Open Source Software Engineering, page Automating the Measurement of Open Source Projects, Portland, Oregon, May 3-10 2003.

[9] T.L. Graves, A.F. Karr, J.S. Marron, and H.Siy. Predicting fault incidence using software change history. IEEE Transactions on Software Engineering, 26(2), 2000.

[10] ToddL. Graves and Audris Mockus. Inferring programmer effort from software databases. In 22nd European Meeting of Statisticians and 7th Vilnius Conference on Probability Theory and Mathematical Statistics, page 334, Vilnius, Lithuania, August 1998.

[11] James Herbsleb and Audris Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In 2003 International Conference on Foundations of Software Engineering, Helsinki, Finland, October 2003. ACM Press.
[12] JamesD. Herbsleb, Audris Mockus, ThomasA. Finholt, and RebeccaE. Grinter. An empirical study of global software development: Distance and speed. In 23nd International Conference on Software Engineering, pages 81-90, Toronto, Canada, May 12-19 2001.
[13] Miryung Kim and David Notkin. Using a clone genealogy extractor for understanding and supporting evolution of code clones. In International Workshop on Mining Software Repositories, 2005.
[14] AnilK. Midha. Software configuration management for the 21st century. Bell Labs Technical Journal, 2(1), Winter 1997.
[15] Audris Mockus. Description and roadmap for a system to measure open source projects. Link to roadmap off http://sourcechange.sourceforge.net/.
[16] Audris Mockus. Analogy based prediction of work item flow in software projects: a case study. In 2003 International Symposium on Empirical Software Engineering, pages 110-119, Rome, Italy, October 2003. ACM Press.
[17] Audris Mockus. Empirical estimates of software availability of deployed systems. In 2006 International Symposium on Empirical Software Engineering, page to appear, Rio de Janeiro, Brazil, September 21-22 2006. ACM Press.
[18] Audris Mockus, RoyT. Fielding, and James Herbsleb. Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology, 11(3):1-38, July 2002.
[19] Audris Mockus and James Herbsleb. Expertise browser: A quantitative approach to identifying expertise. In 2002 International Conference on Software Engineering, pages 503-512, Orlando, Florida, May 19-25 2002. ACM Press.
[20] Audris Mockus and LawrenceG. Votta. Identifying reasons for software change using historic databases. In International Conference on Software Maintenance, pages 120-130, San Jose, California, October 11-14 2000.
[21] Audris Mockus and DavidM. Weiss. Predicting risk of software changes. Bell Labs Technical Journal, 5(2):169-180, April-June 2000.
[22] Audris Mockus and DavidM. Weiss. Globalization by chunking: a quantitative approach. IEEE Software, 18(2):30-37, March 2001.
[23] Audris Mockus, DavidM. Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In 2003 International Conference on Software Engineering, pages 274-284, Portland, Oregon, May 3-10 2003. ACM Press.
[24] Audris Mockus, Ping Zhang, and Paul Li. Drivers for customer perceived software quality. In ICSE 2005, St Louis, Missouri, May 2005. ACM Press.
[25] Rational Software Corporation. ClearCase Manual. May be found on: http://www.rational.com/.
[26] M.J. Rochkind. The source code control system. IEEE Trans. on Software Engineering, 1(4):364-370, 1975.
[27] Annie Ying, Gail Murphy, Raymond Ng, and Mark Chu-Carroll. Predicting source code changes by mining change history. IEEE Transactions of Software Engineering, 30(9), 2004.

Measurement and Interpretation of Productivity and Functional Correctness

Hakan Erdogmus

Measuring developer productivity and functional correctness is central to evaluating software practices and techniques. Researchers use a wide variety of measurement and reporting methods. As such, the interpretation, aggregation and comparison of experimental results become difficult. The problems often reduce to the proper ways of defining units of development work and quantifying developer output. For example, when is it appropriate to measure productivity in terms of elapsed problem solving time vs. output per unit time? Is number of lines of source code an eternally damned output measure in all situations? How do we define task completion to measure problem solving time? What are the consequences of having a cut-off time? When is a minimum quality or usability criterion necessary? How should such a criterion be defined? What are some good output metrics that proxy external functionality? How can we effectively measure these metrics? When is it acceptable to define functional correctness as a binary variable? What are the pros and cons of objective vs. subjective measures?

While defining measures that are universally meaningful and applicable is not feasible, researchers, reviewers, and end readers would benefit from metric selection and reporting guidelines for some common contexts. Selecting proper measures will remain largely a matter of reconciling the several trade-offs involved in experimental design. However, guidelines that explicitly identify these trade-offs may prevent their arbitrary resolution, thus both streamlining and facilitating their interpretation.

Emerging approaches for automated acceptance testing [1] and process telemetry [2] raise some interesting possibilities for the definition, measurement, and selection of quasi-standard measures of productivity and correctness. It would be worthwhile to investigate the possibility of leveraging these developments to assist experimental researchers.

References

1. H. Erdogmus, M. Morisio, M. Torchiano, " On the effectiveness of Test-Driven Development," IEEE Trans. Software Engineering, 31(1), January 2005.
2. P.M. Johnson, et al, "Software Development Management through Software Project Telemetry," IEEE Software, 22(4), July 2005.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 100, 2007.

Synthesising Research Results

Barbara Kitchenham

I am currently working on the Evidence-Based Software Engineering (EBSE) project (EP/C51839X/1) awarded by the UK Engineering and Physical Sciences Research Council. We are investigating how readily the concept of evidence-based practice can be adapted to Software Engineering. Evidence-based practice relies on research synthesis, aggregating and analysing relevant 'primary studies' by using the methodology of systematic literature reviews. Much of the original impetus came from medical research and emphasised synthesis of high-quality quantitative experiments (i.e. double-blind randomised controlled trials) using statistical meta-analysis.

As part of our research we are reviewing the use of evidence-based practice in social science related domains which have empirical practices that are more similar to Software Engineering than those of medicine [1] Research originating in these disciplines has identified sound methods for synthesising evidence from different types of studies, including qualitative studies [2].

In particular, research synthesis of mixed study types needs to:

- Assess the quality of each primary study using criteria appropriate to the study type.
- Use study quality and study type as "moderator variables" to assist interpreting data.
- Synthesise studies based on different methods separately.

This approach can be used for research synthesis of systematic reviews including both qualitative and quantitative primary studies. For example, a cross-study synthesis of .a quantitative systematic review and a qualitative systematic review should assess the extent of agreement between the reviews. Interventions considered important in both types of survey are likely to be the most beneficial.

References

1. David Budgen, Stuart Charters, Mark Turner, Pearl Brereton, Barbara Kitchenham & Steve Linkman. (2006) Transferring the Evidence-based paradigm to Software Engineering. Proceedings of the WISER workshop, ICSE 2006, ACM Press, pp 7-13.
2. Mark Petticrew and Helen Roberts. Systematic Reviews in the Social Sciences. A Practical Guide. Blackwell Publishing Ltd. 2006.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 101, 2007.

On the Quality of Data

Thomas Ostrand

Accurate and reliable data is critical to derive conclusions from software engineering empirical studies. Our experience in collecting data from several large industry projects has taught us that raw data frequently is not what it seems, and that great care must be taken in interpreting information provided by software developers and testers.

A reliable contact person within a project, preferably someone who has been associated with the project for its entire lifetime, is an invaluable resource for describing the project's databases, and explaining how to access them. Such a contact can also provide insight into the project history, explain the project's culture, and frequently resolve the meaning of otherwise obscure entries in the databases.

Online software databases are typically populated both by programs and by human users. Both types of information can be misleading, as shown by the following examples:

- We collected data from 35 consecutive releases of a large system. The version control system automatically recorded the initialization date of each release, which is a key value for our models. It turned out that this recorded start date wasn't necessarily the real start date. Sometimes, the release was entered into the database well before it was actually populated with any files. We ended up using the first date of file population as the release start date.
- The problem report forms have a severity (1-4) associated with each MR. This value is supposed to indicate the importance of the problem, and be a guide to the urgency of fixing it. Unfortunately, we have found that the value chosen can sometimes be based on social reasons, rather than providing a realistic evaluation of the problem's importance.
- Fields in online report forms may have default values; if the user enters nothing, the default is used. If the project doesn't emphasize the importance of the user actually making a choice, this can seriously skew the result totals.

Recommendations for data collection:

- Understand exactly how fields in data collection forms are filled out
  - Are default values supplied if the user doesn't make a choice?
  - If possible, ask the project management to eliminate default values in forms.
  - What are the available options for users?
  - Do users choose from a drop-down list, from a radio list, or do they fill in free text.
- Understand the semantics of values that are provided
  - Fields such as category, phase, and severity may have different meanings to different projects, even within the same company.
  - The meanings of values may change over time even within a single project.
- Validate collected data
  - If the amount of collected data is manageable, examine each individual entry to assess its consistency and reasonableness.
  - If the data set is too large for individual examination, validate a randomly selected subset.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 102, 2007.

Potential of Open Source Systems as Project Repositories for Empirical Studies
Working Group Results

Nachiappan Nagappan

Abstract. This paper is a report on the working group discussion on the potential of open source systems as project repositories for empirical studies. The discussions of this working group focused more generally on the typical questions that can be answered using the open source data, the challenging issues that can be addressed in future, and a discussion on the results to date.

1 Introduction

The working group consisted of the following participants (in alphabetical order): Giovanni Cantone, Audris Mockus (lead), Sandro Morasca, Nachiappan Nagappan, Lutz Prechelt, Giulherme Travassos, Larry Votta.

Open source software repositories contain a wealth of valuable information for empirical studies in software engineering. Unlike most commercial repositories that miss information verbally exchanged by developers, they contain all archived communications among project participants and record the rationale for decisions throughout the life of a project. This completeness is attributable to the fact that email archives, problem tracking systems, and version control systems represent the only way to exchange information among project participants who reside in multiple continents and develop software projects without ever meeting in person.

Several studies have used this data to study various aspects of software development such as software design/architecture, development process, software reuse, and developer motivation. These studies have highlighted the value of collecting and analyzing this data. Yet the methodology and tools to address the challenge of utilizing such data to perform empirical research are complex and not embraced by all researchers.

The discussion in this working group was primarily in two area's, (i) typical open source data sources (ii) challenging issues that can be addressed using open source data and some of the results obtained using open source data. In this discussion we also provide some related avenues for typical research on open source.

2 Open Source Data

Several interesting questions can be discussed using data from open source systems. The open source repositories contain information that is available for public use to facilitate volunteer participation in these projects and to ensure the software can be maintained by anyone having such need and resources. Some of the more commonly available artifacts that can be extracted are,

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 103 – 107, 2007.

- Structural information
  - source code via the version control systems
  - program dependences
- Process
  - test results (for example, Fitnesse)
  - build results from a commonly used tinderbox tool
  - coverage information (for example, jcoverage)
  - bug database (for example, Bugzilla)
  - some project management tools (like feature management tools, effort tracking tools etc.)
- Organizational structure and communication patterns
  - project participant behaviour from their participation in code changes, problem reports, and mailing lists
  - Design decisions and rationale from mailing lists

Such data can be used to generate more exploratory hypotheses. An important point to be noted is the analysis is strongly dependent on what the research(-er) group is willing to invest. A related chapter in this book titled "Software Support Tools and Experimental Work" discusses in detail the tools that support software development and the raw data retrieval. Further, based on the discussions in our working group, joining the project to implement research techniques or test more involved hypotheses not only provides better validation and interpretation of research results but also serves to answer some of the ethical questions of giving back to the open source projects.

3 Research Challenges in Open Source Systems

Spinellis and Szyperski (in 2004) [11] discuss the number of open source projects available to developers. Table 1 summarizes [11] some of the numbers they discuss. This data is not an exclusive sample as the authors identify that many projects appear on more than one of the locations listed. But what is interesting is the sheer number of projects available for empirical study. Since the number of projects available for use is large, researchers should carefully understand the projects used in empirical study. Generalizations are often difficult in this environment. An issue with trying to draw generalizations is the criteria that are to be used to select open source projects.

Table 1. Open source project repositories

Repository	Number of projects
http://freshmeat.net	30,000
http://sourceforge.net (of varying quality, completeness, and stability)	70,000
www.cpan.com (Comprehensive Perl Archive Network)	5400 modules
FreeBSD operating system	10,000 ports all regularly tested are distributed with the FreeBSD system

As an example, in order to predict post-release field quality based on automated unit testing, (Trouble reports – TR) the following criteria was used for selecting projects from an open source repository (Sourceforge) [7].

- *software development tools.* All of the chosen projects are software development tools, i.e. tools that are used to build and test software and to detect defects in software systems. This is to ensure all projects used in the analysis belong to a similar application domain.
- *download ranking of 85% or higher.* In Sourceforge, the projects are all ranked based on their downloads on a percentile scale from 0-100%. For example, a ranking of 85% means that a product is in the top 85% based on the number of downloads. We chose this criterion because we reasoned that a comparative group of projects with similarly high download rates would be more likely to have a similar usage frequency by customers that would ultimately reflect the post-release field quality.
- *automated unit testing.* The projects needed to have automated tests.
- *defect logs available.* The defect log needed to be available for identifying TRs with the date of the TR reported.
- *active fixing of TRs.* The TR fixing rate is used to indicate the system is still in use. The time between the reporting of a TR and the developer fixing it serves as a measure of this factor. Projects that had open TRs that were not assigned to anyone for a period of three months were not considered as they indicated projects that were no longer actively supported.
- Sourceforge development stage of 4 or higher. This denotes the development stage of the project (1-6) where 1 is a planning stage and 6 is a mature phase. We chose a cut-off of 4 which indicates the project is at least a successful beta release. This criteria indicates that the projects that are at a similar stage of development and are not projects too early in the development lifecycle.

The working group discussions also involved the issue of scale. Techniques that don't scale well may still be the best choice for investigating smaller open source projects. Some interesting research questions that could be investigated using open source systems that came up during the working group discussion were,

- Comparing programming languages, e.g. C vs Java
- Evaluating build, test tools, process
- Do software assertions work? Can we leverage the JUnit data there is with problem reports
- What is the nature of social interactions in an open source environment
- How does the global distribution of developers affect the process? What are the needs in terms of tools, techniques and processes?

Further, in this section we also discuss recent research using open source systems to server as starting pointers for further investigation. Table 2 provides such a description based on the systems analyzed.

Table 2. Example recent work on open source systems

Open source system	Example research literature description
Mozilla	Email archives of source code change history and problem reports are used to quantify aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution intervals (Mozilla and Apache) [6].
Apache	Estimating fault-proneness for Apache 2.0 using software metrics data from Apache 1.3 [3].
Games (e.g. Quake)	software development practices, social processes, technical system configurations, organizational contexts, and interrelationships [9].
Linux	Investigates the extent and the evolution of code duplications in the Linux kernel [1].
FreeBSD	Understanding the open source process for FreeBSD [4]
Eclipse	Investigate if import dependences can predict failures [10]
PostgreSQL	Investigates large scale software evolution using linker-based extraction [12].
OpenBSD	Predicting field defects using software reliability growth models and metrics based modeling approach [5].

4 Conclusions

Open source software systems provide a tremendous public repository of data that can be mined to understand and provide insights into the overall software process, tools and social interactions related to software development in general. But as with all empirical studies in software engineering drawing general conclusions from empirical studies in software engineering is difficult because any process depends to a large

Table 3. Example venues for open source systems work

Resource	Description
International Conference on Open Source Systems (OSS)	http://oss2005.case.unibz.it/ . The OSS2005 conference focused (i) the development of open source systems and the underlying technical, social, and economics issues and (ii) the adoption of OSS solutions and the implications of such adoption both in the public and the private sector.
PROMISE Software Engineering Repository [8]	http://promise.site.uottawa.ca/SERepository/ . This repository consists of publicly available datasets and tools to serve researchers to build predictive models.
Mining Software Repositories workshop	http://msr.uwaterloo.ca/msr2006/ . The goal of this workshop is to establish a community of researchers and practitioners who are working to recover and use the data stored in software repositories to further understanding of software development practices.

degree on a potentially large number of relevant context variables. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment in which it was conducted [2]. Researchers should understand the complexities and the intricacies involved in the development process while mining the open source repositories. There is always a danger of misinterpreting the data as it is easy to extract and publicly available, yet the interviews with project participants needed to verify the methodology and the interpretation of results may be hard or impossible to obtain. In the coming years studies using open source systems can provide great insight into software development and to software engineering in general. (Table 3 lists some example venues for open source data and related scientific events).

References

[1] G. Antoniol, Villano, U., Merlo, E., Di Penta, M., "Analyzing cloning evolution in the Linux kernel", *Information and Software Technology*, 44(13), pp. 755-765, 2002.

[2] V. Basili, Shull, F.,Lanubile, F., "Building Knowledge through Families of Experiments", *IEEE Transactions on Software Engineering*, 25(4), pp. 456-473, 1999.

[3] G. Denaro, Pezze., M, "An empirical evaluation of fault-proneness models", Proceedings of International Conference on Software Engineering, pp. 241-251, 2002.

[4] T. T. Dinh-Trong, Bieman, J., "The FreeBSD Project: A Replication Case Study of Open Source Development", *IEEE Transactions in Software Engineering*, 31(6), pp. 481 - 494, 2005.

[5] P. L. Li, Herbsleb, J., Shaw, M., "Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of OpenBSD", Proceedings of International Symposium on Software Reliability Engineering, pp. 193-202, 2005.

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

[7] N. Nagappan, "A Software Testing and Reliability Early Warning (STREW) Metric Suite," in *Computer Science Department*. PhD Thesis, Raleigh: North Carolina State University, 2005.

[8] J. Sayyad Shirabad, Menzies, T.J., "The PROMISE Repository of Software Engineering Databases," School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[9] W. Scacchi, "Free and Open Source Development Practices in the Game Community", *IEEE Software*, 21(1), pp. 59-66, 2004.

[10] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting Component Failures at Design Time," in International Symposium on Empirical Software Engineering, 2006.

[11] D. Spinellis, Szyperski, C., "How is open source affecting software development?" *IEEE Software*, 21(1), pp. 28- 33, 2004.

[12] J. Wu, Holt., R.C., "Linker-Based Program Extraction and Its Uses in Studying Software Evolution", Proceedings of International Workshop on Unanticipated Software Evolution, pp. 1-15, 2004.

Data Sharing Enabling Technologies
Working Group Results

Marvin V. Zelkowitz

Attendees: M. Zelkowitz (chair), V. Basili, R. Glass, M. Host, M. Mueller, T. Ostrand, A. Rainer, C. Seaman and H. Sharp

Issues

If empirical software engineering is to prosper as a research domain, the field needs a mechanism for the sharing of data and artifacts developed by one group that may be useful to another group desiring to work in the same area. In the opening session, the workshop discussed a forthcoming position paper entitled "Protocols in the use of Empirical Software Engineering Artifacts" [1]. This paper describes a taxonomy of properties that are necessary in order to appropriately share information. These properties include who has *permission* to use this data, what *protection* (e.g., privacy) is given to the subjects who provided the data, what *credit* does the user of this data owe the provider of the data, what are the roles governing joint *collaboration* of the activity, who must *maintain* the integrity and access to the data, and what *feedback* does the user of the data owe the creator of the data? The breakout session discussed this taxonomy is greater detail focusing on both qualitative and quantitative data and on what enabling technologies were needed in order to further advance the needs for data ownership and sharing.

The data that is shared can be broken down into 4 classes of artifacts:

1. *Quantitative data* that is the result of measuring an activity. In the software development domain this usually means time data (effort in hours or calendar dates), error data (number and types of defects), and product data (names and sizes of components, execution times, results of testing, etc.)
2. *Artifacts produced* such as source code, design documents, test plans, etc.
3. *Tools* needed to collect data, such as test harnesses, data collection tools, such as Hackystat, etc.
4. *Procedures* for collecting data, such as reporting forms, requirements documents, provided test data, etc.

Qualitative data generally consists of the latter 3 classes. The collected data is often the artifact that is produced. In many cases, such as with ethnographic studies, the collected data consists of interview notes, or tape and video recordings of the subjects of the study.

For both classes of data (i.e., quantitative and qualitative) the proposed taxonomy [1] is still incomplete and still needs additional work in the areas of:

1. *Addressing the proper context of the data*. Understanding the work environment and the proposed application domain for the product under study plays an important role in understanding what the data means.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 108 – 110, 2007.

2. *Understanding common terminology*. Before any meaningful discussion on data sharing can occur, there needs to be a shared understanding of what the objects under study are. During this workshop, three speakers all referenced taxonomies for classifying research methods, and all were different [2, 3, 4].
3. *Need agreement with subjects of what is owned by each*. For example, if data represents a developed program, who owns the rights to that program? Do students own the results of their programming, does the university where the student is enrolled own those programs, or does the research group using the data own those programs? This still is an unresolved issue.
4. *Adherence to national standards and laws*. Related to the previous ownership issue is the fact that rules differ by locality. Various national laws exist governing the collection and dissemination of collected data. It is not obvious how to generate a useful taxonomy usable by the worldwide empirical software engineering community that meets all national and international regulations.
5. *Ethics*. With ownership comes responsibility. What are the obligations and responsibilities of the owner of the data to ensure that the data is used correctly and what are the obligations and responsibilities of the user of the data that results are provided using the proper context for the data?
6. *Provenance*. While maintenance and integrity of the data has already been identified in the taxonomy, the issue of provenance has not been separately identified. As a data set evolves over time, and it will if it represents useful data, then the set of artifacts must be traceable back to their origins and all data must also be accounted for in an unbroken string from its creation to its eventual use.

Enabling Technologies

The focus of the breakout session was to define a roadmap of what enabling technologies, and related research, were needed in order for the data sharing concept to become accepted by the empirical software engineering community. The underlying principle for acceptance of this concept was that "use breeds additional use." If a successful taxonomy can be developed and used by much of the empirical software engineering community, then the rest will follow along. So the issue was how to get an acceptable policy acceptable to most in the field?

The first step is to give the proposed taxonomy wide distribution, with requests for comments and feedback. Copies of the paper [1] were distributed to all workshop attendees and the paper was submitted for publication in *Empirical Software Engineering*. The paper is also scheduled for discussion at the September 2006 International Software Engineering Research Network (ISERN[1]) meeting in Rio de Janeiro, Brazil.

Two other models of sharing were also discussed.

1. *SourceForge.net.*
    a. SourceForge.net, a widely used repository of open source software, maintains a licensing agreement for individuals wanting to use or modify SourceForge products. The success of that licensing policy needs to be studied as an indicator of the problems and issues our empirical software engineering data sharing policy will face.

[1] http://www.cos.ufrj.br/~ght/isern2006.htm

   b. The open source community, as represented by SourceForge, seems to have developed a viable economic model that includes free access to the software. For example, the basic system is free, but add-on features cost extra. This is a major issue in the empirical software engineering data sharing process. We would like data to remain viable for many years, yet it takes funding to maintain the databases. Universities do not have that source of funds and funding agencies are not willing to fund these activities. Perhaps the Open Source model provides a solution.
2. A Material Transfer Agreement (MTA) is a contract that governs the transfer of tangible research materials between two organizations. The MTA defines the rights of the provider and the recipient with respect to the materials and any derivatives. While biological materials are the most commonly transferred items, MTAs may also be used for other types of materials, such as some types of software.

A proposal was discussed covering the ethics issues mentioned earlier. Any paper submitted to a journal using data from an existing source would have an additional review by the creator of that data to ensure that the data was used correctly. This additional review would not be an accept/reject decision (after all, the new paper may correctly say something negative about the original data source, which the additional reviewer may not appreciate), but would provide the editor with additional non-binding comments about the appropriateness of the data analysis used in the paper.

In addition, it was discussed that conferences and journals should provide a small amount of space (and time at a conference) to describe a data source so that others interested in using that data has the opportunity to learn about it and secure a copy. The paper should describe the data set, what it contains, the various constraints in the data, and various experiences that others have had in using the data. The more these are described at meetings, the bigger the market will grow in using data.

We have anecdotal information that advertising data sets do get them used. In revising the 1998 survey on validating computer technology [4] to include data from the years 2000 and 2005 for this workshop, it was noticed that the number of papers using existing data sets greatly increased over the 1998 survey. Most of this increase was in papers using Open Source libraries for such products as Eclipse, the Apache web serve and the Mozilla browser. Given a reliable source of data, researchers are very willing to obtain it for their own research. Formalizing the process with an evaluated taxonomy can only help the process of increasing the supply of good experimental data.

References

1. V. Basili, M. Zelkowitz, D. Sjøberg, P. Johnson and T. Cowling, Protocols in the use of Empirical Software Engineering Artifacts (submitted for publication).
2. D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N-K Liborg and A. C. Rekdal, A survey of controlled experiments in software engineering, *IEEE Trans. on Soft. Eng.* 31, 9 (2005) 733-753.
3. W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, Experimental evaluation in computer science: A quantitative study, *J. of Systems and Software* 28, 1 (1995) 9-18.
4. M. V. Zelkowitz. and D. Wallace, Experimental models for validating computer technology, *IEEE Computer* 31, 5 (May, 1998) 23-31.

Documenting Theories
Working Group Results

Dag I.K. Sjøberg

Attendees. Marcus Ciolkowski, Tore Dybå, Frank Houdek, Andreas Jedlitschka, Barbara Kitchenham, Dietmar Pfahl, Dag Sjøberg (chair), Sira Vegas and Laurie Williams

1 The Need for Theories

There are many arguments in favour of theory use, such as structuring, conciseness, precision, parsimony, abstraction, generalisation, conceptualisation and communication [1,9,11]. Such arguments have been voiced in the software engineering community as well [2,3,8,10]. Theory provides explanations and understanding in terms of basic concepts and underlying mechanisms, which constitute an important counterpart to knowledge of passing trends and their manifestations. When developing better software engineering technology for long-lived industrial needs, building theory is a means to go beyond the mere observation of phenomena, and to try to understand *why* and *how* these phenomena occur.

The issue of this workshop was how empirically-based theories in software engineering should be documented. Since some work in this area had already been reported by some of the attendees in the paper [5], the highlights of that paper, including Table 1, were presented using a projector. (The paper was sent by email to the attendees after the workshop.) A graph was also presented that showed which controlled experiments in software engineering used which theories in what ways.

It was suggested that it would be useful with information about which theories were used in other study types as well, not only experiments. Since such systematic reviews are formidable jobs, it was suggested to first focus on certain sub-areas of software engineering. One attendee said he would start collecting the theories used within requirements analysis.

2 Schemes for Documenting Theories

Table 1 shows the scheme used to document theories in the systematic review reported in [5]. The general feeling of the attendees was that this scheme seemed useful.

It was then shown how a web site in the information systems field collects and documents theories [7]. The scheme used is shown in Table 2. It was discussed whether one would like a similar web page within software engineering. The attendees were positive, but some people in the plenary session were critical when the summary from this workshop was reported afterwards. It was argued that there was a risk that one would collect many useless theories. Addressing this issue would require a careful inclusion acceptance procedure. Table 3 shows the guidelines that are used

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 111 – 114, 2007.

Table 1. Attributes of theory used in [5]

Metadata
- *Name*. The name given for the theory by the authors of the reviewed article, or by us if no explicit name is given.
- *References*. The literary references given for the theory.
- *Terminology* (*theory, model, none*). Indicates explicit use of the word 'theory', 'model' and their derivatives in referring to the theory.
- *Reference discipline*. The discipline(s) of an article's literary sources to the theory.
- *Topic*. The topic of the article in which the theory is used.

Structural components – Generic (adapted from [4])
- *Means of representation* (*words, tables, diagrams, mathematics, logic*). The way the theory is presented.
- *Constructs and relationships*. Examples of main constructs and relationships found for the theory.
- *Boundary conditions*. Indications given of the theory's boundary conditions.

Structural components – Contingent on theory type (adapted from [4])
- *Causal explanations*. Indications that the theory gives statements of relationships among phenomena that show causal reasoning (not covering law or probabilistic reasoning alone).
- *Predictions*. Indications that the theory gives statements of relationships between constructs in such a form that their operationalisations can be tested empirically.
- *Prescriptive statements*. Indications that the theory gives statements that specify how people can accomplish something in practice, *e.g.,* construct an artefact or develop a strategy.

Theory role:
- *design*: the experiment's research questions and hypotheses are justified or motivated by the theory
- *post hoc explanation*: the theory is used after the experiment to explain observed phenomena
- *tested*: the theory is tested by the experiment – *derivation*: derivation of theory, *instance*: instance of theory
- *modified*: the theory is enhanced, refined, conditionalised, *etc.* based on the experimental findings
- *proposed*: a major part of the theory is proposed by the author(s) in the current reviewed article, and the theory is used in one of the preceding roles or is based on treatment-outcome relations of the experiment
- *basis*: the theory is used as a basis for other theory used in one of the preceding roles.

Calls for Theory: Records any calls for theory or comments on the lack of theory being problematic.

for contributors to the IS theory web site. This seems like a good starting point for guidelines to contributors to a theory web site for software engineering. Hence, Simula Research Laboratory has began building a similar site for SE theories and has initiated collaboration with those managing the IS theory web site.

Table 2. Attributes of theories used in the IS Field dataset [7]

- Name of theory
- Acronym
- Alternate name(s)
- Main dependent construct(s)/factor(s)/variable(s)
- Main independent construct(s)/factor(s)/variable(s)
- Concise description of theory (max 500 words)
- Diagram/schematic of theory
- Originating author(s)
- Originating article(s) - capture in separate worksheet/document
- Originating area (economics, psychology, etc)
- Level of analysis (individual, group, firm, industry, etc.)
- Hyperlinks to WWW sites describing theory with brief description of link
- Links from this theory to other theories
- IS articles that use theory - capture in separate worksheet/document
- Contributor(s)
- Date of latest summary

Table 3. Guidelines for contributors to the IS Field dataset [7]

1. Contributors of new theory summaries should use the format of present summarized theories and electronically submit the text to the editors using this template.
2. New summaries should be complete; summaries missing significant portions will not be accepted.
3. Corrections or additions to existing theory summaries are welcomed, and should be justified to the editors.
4. References should be in the standard MISQ reference format: (http://www.misq.org/roadmap/standards.html).
5. All contributors agree that any information submitted to this site becomes the property of the site and is presented publicly under a GNU Free Documentation License.
6. Contributions will be acknowledged by email, reviewed by the editors, and may be forwarded to the site advisors for comment. Accepted contributions will, in due course, be posted on the site with attribution to the contributor.

Table 4. Attributes of theories in psychology collected at [6]

- Description
- Example
- So what?
- Using it
- Defending
- See also (other theories)
- References

Another web site to look at is one that records theories in psychology [6]. Their format of theory description is relatively simple, see Table 4. This scheme can be compared with the scheme of the IS web site by looking at, for example, the cognitive dissonance theory, which is described at both sites. The workshop did not discuss in detail what kind of scheme would be appropriate in a possible theory web site for

software engineering, but the schemes shown in Tables 1, 2 and 4 are certainly good starting points.

References

1. S.B. Bacharach. Organizational theories: Some criteria for evaluation. Academy of Management Review, 14(4):496–515, 1989.
2. V.R. Basili. Editorial. Empirical Software Engineering, 1(2), 1996.
3. A. Endres and D. Rombach. A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories. Fraunhofer IESE Series on Software Engineering. Pearson Education Limited, 2003.
4. S. Gregor. The nature of theory in information systems. MIS Quarterly, 30(3) : 611–642, 2006.
5. Jo E. Hannay, D.I.K. Sjøberg, T. Dybå, A Systematic Review of Theory Use in Software Engineering Experiments, IEEE Transaction on Software Engineering, 33(2) : 87–107, February 2007.
6. http://changingminds.org/explanations/theories/theories.htm
7. http://www.istheory.yorku.ca/
8. B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. IEEE Transaction on Software Engineering, 28(8):721–734, August 2002.
9. J.W. Lucas. Theory-testing, generalization, and the problem of external validity. Sociological Theory, 21:236–253, 2003.
10. W.F. Tichy. Should computer scientist experiment more? 16 excuses to avoid experimentation. IEEE Computer, 31(5):32–40, May 1998.
11. D.G. Wagner. The growth of theories. In M. Foschi and E.J. Lawler, editors, Group Processes, pages 25–42. Nelson–Hall Publishers, Chicago, 1994.

Measurement and Model Building
Discussion and Summary

Sira Vegas and Vic Basili

1 Introduction

This chapter summarizes the discussions that took place during the *Measurement and Model Building* session of the Dagstuhl seminar on Empirical Software Engineering (ESE). The goal of this session was to address questions concerning two topics: data sharing and effective data interpretation.

The chapter has been organized as follows. Section 2 presents the discussions during the data sharing presentations. Section 3 deals with the discussions of effective data interpretation presentations. Section 4 summarizes the topics that were discussed by parallel groups after the presentations. Section 5 sums up the session outcomes.

2 Data Sharing

This section reviews the discussions that took place during Dag Sjøberg's and Richard Selby's presentations concerning data sharing. These two talks are detailed in the *Knowledge Acquisition in Software Engineering Requires Sharing of Data and Artifacts* and *Data Collection, Analysis, and Sharing Strategies for Enabling Software Measurement and Model Building* chapters of this book, respectively.

2.1 Knowledge Acquisition in Software Engineering Requires Sharing of Data and Artifacts (by Dag Sjøberg)

Building bodies of knowledge by accumulating knowledge from empirical studies requires the sustained efforts of several research groups. To achieve this goal, experimental artifacts have to be shared. However, making material accessible to others may require substantial effort by the creator. How should this creator benefit from such an effort, and how should the likelihood of misuse be reduced to a minimum? This talk explores these issues.

This presentation prompted discussion of the following topics: the definition of the term theory, replications by the same vs. different researchers, why a license is needed, who are the owners of the data, and license expiration.

- *Definition of the term theory.* The term theory is not yet well understood by the community, as illustrated by the fact that several people in the audience asked for an example.
- *Replications by the same vs. different researchers.* A study has shown that more confirmatory results are obtained when a study is replicated by the same researchers than when it is replicated by different ones. The speaker pointed out the possibility of a bias. However, there is no agreement with respect to this issue as

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 115 – 120, 2007.

some people in the audience remarked that it could be due to human interaction or implicit context variables that differ.

- *Why a license is needed.* There is no agreement in the community on whether a license is really needed. According to the people who are in favor of licensing, the license proposal is motivated by several issues: wanting to know who is using the data artifacts, and any results obtained from its use, the possibility of some researchers misinterpreting or misusing the data without support from the original researchers. However, one suggestion was that we use the Open Source (OS) model about sharing data and artifacts. The risk of misinterpretation or misuse also exists in OS, and this does not stop them from being open.

  Another suggestion was that we should look at what people in physics and medicine do. But a follow up comment was that each field has its own particular problems and issues and we have to tailor any such rules to the ESE field.
- *Who the owners of the data are.* The speaker questioned researchers' ownership of data. However, it is not clear who the real owners of the data are, as different people suggest different options. Should it be the people who provide researchers with the data? The organization these people belong to? The experimental subjects? The government in the case of government-funded projects?
- *Maintaining the data and artifacts.* The issue was raised that data storage, retrieval, evolution costs are real and there is no source of funding that allows data and artifacts to be maintained, quality assured, etc.
- *License expiration.* Somebody in the audience brought up the timing issue. For how long can the people who borrow the data use it?

2.2 Data Collection, Analysis, and Sharing Strategies for Enabling Software Measurement and Model Building (by Richard Selby)

Successful software measurement and model building require effective strategies for data collection, analysis, and sharing. This talk presents a template for data sharing arrangements, along with its application to two environments.

The discussion during this presentation focused on the following points: why data sharing is a problem, its scope, and students as owners of data. Some issues had already been raised in Dag's talk, and they are discussed here in more detail, specifically the issue of data sharing and data ownership.

- *The problem of data sharing.* Some participants did not understand why data sharing is a problem. It was explained that data sharing entails a number of problems. One of them is that it is very costly to maintain data (and the owners have to maintain it). Another one is that some data is private and cannot be shared. This is the case of the data gathered from a company. To illustrate the problem, two examples were given. The first example was that years ago it was possible to use various forms of student programs obtained for experiments, but now these programs are considered the intellectual property of the student and cannot be easily shared. The second example was that the data used to develop various models like COCOMO are often company owned and cannot be shared.

It was also explained that the point is not to obstruct the work of the people who are borrowing the data, but to restrict what they can do with it, and acknowledge the people who own it.

However, some people questioned how real the problem is. They believe that researchers are usually willing to share their data, e.g. the NASA SEL shared its data via a database at the Rome Air Development Center DACS.

- *What the solution is.* Some people doubted that restricting what can be done with the data or acknowledging the people who own it would motivate data owners to lend it. Other proposals were made, like co-authorship of papers (to reflect joint work).
- *Scope of the problem.* Another issue that came up was whether sharing issues affect just data or also extend to any kind of artifact used to run empirical studies.
- *Students as owners of data.* Some people believe that when running experiments with students, they are the owners of their data. In that particular case, we should ask them whether they mind the whole ESE community using their data.

3 Effective Data Interpretation

This section summarizes the discussions that took place during Jürgen Münch's and Audris Mockus' presentations about data interpretation. These two talks are detailed in the *Effective Data Interpretation* and *Software Support Tools and Experimental Work* chapters of this book, respectively.

3.1 Effective Data Interpretation (by Jürgen Münch)

Drawing useful conclusions from individual empirical studies and combining results from multiple studies requires sound and effective data interpretation mechanisms. This talk sketches the progress made in data interpretation in the last few years and presents needs and challenges for advanced data interpretation.

Only one discussion topic came up in this presentation. It focused on:

- *Integrating project monitoring into empirical studies.* The results obtained from project monitoring activities cannot serve as a substitute for running empirical studies. However, project monitoring can be used to help interpret the results of empirical studies. The data from the empirical studies can be used to calibrate models and run simulations.

3.2 Software Support Tools and Experimental Work (by Audris Mockus)

Using software support tool repositories to explain and predict phenomena in software projects is a promising idea. This presentation outlines the opportunities and challenges of using project support systems in empirical work.

The discussion in this presentation focused on understanding data in OS repositories and the role of OS in ESE:

- *Understanding data in repositories.* Although OS repositories are a potential source of data, somebody asked whether it is possible to understand this data. The

answer was that e-mails are particularly useful in these projects. OS people do not sit and talk, but they do everything via e-mail. Therefore, a lot of information can be gathered from e-mails. However, you still need to talk to them, although less so than in closed environments.

- *The role of OS in Empirical Software Engineering.* Somebody asked if it was being suggested that future empirical studies should be derived from OS projects. The answer was no. It is just that OS repositories are a rich source of data, often containing more information than closed environments.

4 Discussion and Summary

The topics proposed for further investigation in parallel working groups were:

- Potential of OS systems as project repositories for empirical studies.
- Documenting theories.
- Data sharing enabling technologies.
- Licensing and sharing issues of qualitative data.
- Prescribing some sort of format for project data sets.

After the voting, the fourth topic was merged with the third one, and the fifth was omitted. This led to the formation of three parallel working groups.

4.1 Potential of OS Systems as Project Repositories for Empirical Studies

The results from the working group examining OS systems as potential project repositories were presented by Audris Mockus and are summarized in the *Potential of OS Systems as Project Repositories for Empirical Studies* chapter in this book.

Only one discussion topic was raised during this presentation, related to the role of the empiricist in the OS project.

- *The role of the empiricist in the OS project.* It was originally suggested that an empiricist should join an OS project as a regular team member. However, somebody asked whether (s)he should not join simply in the role of an empiricist instead. It is possible to join the team as an empiricist as long as you do not bother people with too many questions. But the reason for joining as a team member is that there are always outstanding tasks that an empiricist can easily do (documentation issues, etc.) to help the team out. This way the empiricist is not viewed as an intruder.

4.2 Data Sharing Enabling Technologies

The results of the working group looking at sharing issues were presented by Marvin Zelkowitz and are summarized in the *Ways to share data* chapter in this book.

There was no discussion in the strict sense during this presentation. However, two remarks were made:

- VTT will make all project-related information (including videos, etc.) available via web (for six months)

- It was suggested that this topic should be added to the next ISERN agenda, and the suggestion was accepted.

4.3 Documenting Theories

The results from the working group investigating theory-related questions were presented by Dag Sjøberg and are summarized in the *Documenting theories* chapter in this book.

The topics of discussion raised during this presentation were related to: documenting a theory and the scope a theory should have:

- *Documenting a theory.* Currently there are a lot of theories, but many are still missing. We should know not only what our own theories are, but also how to document them. The http://www.istheory.yorku.ca/ web page was presented as an example of the contents of a theory. The question is whether we should use a similar format. There was agreement on the usefulness of the table associated with each theory, but some people disagreed on whether we could represent our theories using the same format. There was also agreement that we need a way to represent what we currently know, and the table could be used as a guide.
- *Scope of theories.* Another problem is how to articulate theories with respect to their scope. Some people agree that current theories are too specific. Unless you narrow the task, they are useless. These specific theories are not useful for generic phenomena. On the other hand, theories that are generic turn out to be vague. The point is that we should be able to break down theories to the level of abstraction necessary to avoid misfit. Perhaps we are mixing the generic with the specific in our theories. Therefore, we need to examine our theories one by one and decide whether or not they are useful. Somebody pointed out that vague theories are also useful, as they can help to unify terminology.

5 Conclusions

By way of a conclusion of this session, we are going to present the main achievements, key points of dissent and key issues to be solved in the coming years. They are summarized in Table 1.

The **main achievements** identified are related to the findings about the need to share data and artifacts to build bodies of knowledge and theories, OS as an opportunity to study project data under certain circumstances and theories as a way of integrating and motivating empirical studies.

The **key points of dissent** identified during the session are related to the data ownership topic. Part of the community does not believe it is a real problem, and therefore they do not see the usefulness of licensing. Finally, there is no agreement on who the owners of the data are.

Finally, several **key issues to be solved in the coming years** have been identified. They are related to the definition, scope and documentation of theories, how to get

Table 1. Conclusions from the session

	TOPIC	EXPLANATION
Main achievements	Share data and artifacts	We need to share data and artifacts to build bodies of knowledge and theories
	Use Open Source	OS is an opportunity to study project data when there is a good email trail
	Build theories	Theories offer a way of integrating and motivating empirical studies
Key points of dissent	Data sharing	Is it a real problem?
	Data ownership	Who are the owners of the data?
	Licensing	Are licenses really needed?
Key issues to be solved in the coming years	Theories	Definition, scope and documentation
	Replication	How to get confirmatory results in replications by different researchers
	Data ownership	How to acknowledge ownership
	Understanding data	Describe data in a repository
	Maintaining data/artifacts	How do we find the funding to maintain data and artifacts for sharing?

successful replications, how to acknowledge owners of data in a paper, how to describe the data contained in a repository so that it can be easily understood by everyone who uses it and, finally, how to find the funding to maintain data and artifacts for sharing.

Technology Transfer and Education
Introduction

Kurt Schneider

Abstract. This session discusses two aspects of making an impact with empirical software engineering results. Improving industrial practices will lead to an obvious, economic benefit. In order to achieve that, researchers will need to partner with practitioners. Empirical software engineering also needs to be integrated into computer science education in order to make an impact later. This session is about all aspects of raising good research questions, and making sure the answers are exploited in practice.

1 Impact of Empirical Software Engineering

The final purpose of empirical software engineering research is to improve industrial practice. A company can achieve that effect by adopting research results, and researchers will be happy to cooperate.

What may seem straight-forward is not so easy in reality. This session addresses issues of identifying good research topics, matching company needs with research agendas, and using education as an important bridge between the two camps. There are several ways to make an impact with empirical research.

1.1 A Discipline in Its Adolescence Must Learn to Reflect

Software engineering is still a rather young discipline. It was first proclaimed in 1968 at the famous NATO Science Committee meeting in Garmisch-Patenkirchen. Even compared to Computer Science, software engineering is still in its adolescence. Like human teenagers, software engineering is pushing forward to create more and more complex software systems. Some of the achievements during that period are highly remarkable. The young discipline pushes forward and shows only a limited desire to reflect on what it has constructed and done so far.

Despite the growing power of software development techniques, uncertainty about many methods and techniques has also grown. How does a company know what testing strategy to follow? Is object-orientation appropriate for a certain kind of embedded software systems? Is it wise to invest in agile practices and let programmers work in pairs? Many questions are highly interesting for researchers to discuss. Industrial decision makers, on the other hand, need answers for their work.

Empirical software engineering has its biggest impact when it is able to answer relevant questions in practice and, thus, directly improve the practice of software development. In our software-dependent world, this contribution will create economic benefit and support our society.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 121 – 124, 2007.

1.2 Matching Interests

A number of prerequisite hurdles need to be taken before we enter into the ideal scenario described above. The order of the steps may vary.

1. Identify crucial questions for industry
2. Make research aware of those questions and their significance
3. Consider questions studied by researchers on their own initiative
4. Match industry problems and research agendas
5. Filter out those questions that can effectively be studied using empirical methods
6. Carry out effective experiments and studies
7. Report findings in a way suitable for practitioners and decision makers
8. Decision makers appreciate and adopt the findings to practical implementation

Fig. 1. Technology transfer as a cooperative process

This lengthy list sheds some light on the issue of "technology transfer" of empirical software engineering research. For the purpose of this session, **technology transfer** is defined as *any process or practice that helps to adopt research results in practice*.

"Practice" means real projects in a company. *Future* research results are also included in this definition; technology transfer is not necessarily a one-way flow of knowledge and innovation, and the intended research will often not be finished when technology transfer starts.

In Figure 1, most steps refer to establishing and using the relationship between two partners. The above-mentioned steps sketch one possible process to bridge that gap. It is dominated by industrial demands and can, thus, be called "pull" technology transfer. Step 3 indicates the alternative of "push" technology transfer in which research offers results to industry. In many real-world situations, push and pull aspects will be interwoven. Technology transfer will often be more complex than indicated by the sketch above. Step 6 refers to the laborious work of planning, setting up, and carrying out empirical studies. But also steps 7 and 8 are important and need to be taken seriously: The answers to good research questions need to find their way back to practice. And it is a common task to "pave that way back to practice" – all hurdles of acceptance need to be removed.

1.3 Education as Long-Term Technology Transfer

Beyond improving industrial practices directly, empirical results can improve them *indirectly*: by improving education of future software engineers. From the viewpoint of universities and colleges, high-quality education is an important goal in itself. This session explicitly included education as the second road to empirical software research impact. In 1992, a corresponding session was entitled "technology transfer" only, but M. Lehman [1] made it perfectly clear that "Software development has traditionally been treated as a people intensive activity. Thus even when potential benefit is demonstrated, management will oppose moves to a capital intensive approach. Until educated otherwise they will resist change and investment, despite the

general trend to industrial mechanisation." This observation still holds. There are good reasons to combine education and technology transfer in one session.

2 Technology Transfer of Empirical Software Engineering Results

Many publications of research results are aimed at the research community. Others are more directed to practitioners. In a written publication, results stand for themselves. When results are supposed to be "transferred" to industry, they need to be revised. The presentation and "packaging" of results has been an aspect studied by many [2, 3].

The organizers consider the following questions crucial for a better understanding of enabling factors. They determine the opportunities of technology transfer (1) of empirical results and (2) of new technologies that were studied by empirical results:

- How do we package results for different purposes and contexts?
- How can we use empirical software engineering to speed up technology transfer?
- What information do practitioners need from empirical studies?
- What kind of presentation is suitable for practitioners?

2.1 Technology Transfer by People Transfer

In a classical example of technology transfer, a student or researcher carries out an empirical study at a university; the study is initiated or sponsored by a company. When that student graduates or the researcher quits and is hired by that same company, there is a smooth and highly individual technology transfer: no need to package material or convince adopters. Education of students turns into a vital prerequisite of technology transfer. This leads to our second topic in this session: How can empirical work impact education?

3 Empirical Software Engineering and Education

Software engineering is still not sure about the impact of many techniques and methods. Empirical results can help to clarify things, and to teach students about new insights. Better-educated students will spread the knowledge in future projects, and in industry.

We need a way to effectively spread what we learn during empirical studies. This raises a number of questions.

- Who will carry out empirical studies?
- How can we teach empirical methods to software engineering students?
- How can we integrate empirical results in our teaching and our curricula? What are approaches and experiences?
- What educational methods are suitable for teaching empirical or evidence-based software engineering?

Software engineering is perceived as a constructive engineering discipline. Students often need to be convinced of the benefit of empirical work. On the other hand, engineering disciplines are used to refer to experiments and studies when they recommend one technique over another.

4 Making an Impact

Good education is a value in itself. Integrating courses on empirical methods into software engineering curricula is a good way to make an impact through education. In the long run, this will also impact industrial practice. However, empirical software engineering also needs faster ways to transfer its results into practice. A good relationship between an industrial and a research partner will facilitate this transfer.

References

1. Lehman, M.M. (1992): Position Paper. In D. Rombach, R. Basili, R.W. Selby (Eds.): Experimental Software Engineering Issues: Critical Assessment and Future Directions. International Workshop Dagstuhl Castle, September 1992. LNCS 706
2. Birk, A., Kröschel, F. (1999): A Knowledge Management Lifecycle for Experience Packages on Software Engineering Technologies. Workshop on Learning Software Organizations, Kaiserslautern, Germany; pp 115-126
3. Houdek, F., Kempter, H. (1997): Quality patterns - An approach to packaging software engineering experience. Symposium of Software Reusability (SSR'97). In: Harandi, M. (Ed.): Software Engineering Notes vol. 22 no. 3, pp 81-88

Empirical Studies as a Basis for Technology Transfer

Elaine J. Weyuker

1 Introduction

The ultimate goal of all software engineering research should be to have an impact on the way software is engineered. This might involve making it more dependable, more economical to produce, more easily understood, or in some way improve the production or quality of software. This should be true whether the research is theoretical in nature or more pragmatic. In the former case the timescale for impact will likely be longer than one would expect if the research involves the proposal of a new specification, development, architecture or testing technique, or a new metric or way of assessing some software artifact. Nonetheless, it should ultimately allow practitioners to build "better" systems.

The primary question we consider in this note is how to effect or facilitate that impact. More specifically, how should we transfer new technology introduced in research to the practitioners who are actually involved in some stage of the development of software systems?

Our experience is that empirical studies are among the best ways to demonstrate to practitioners how the technology works or is to be used, and to convince practitioners that the technology will actually be usable by their project. This, of course, assumes that the empirical studies are believable and relevant to the types of projects that the practitioners are actually building. For this to be the case, it is generally necessary to perform these studies using "real" projects. The question then becomes, how do we get projects to participate, and how do we distribute the results so that the technology eventually gets transferred? These two issues are the subject of this paper.

2 Finding Candidates for Case Studies and Performing Them

Not surprisingly, we have observed that it is most difficult to find the initial project to participate in an empirical study. Some of the strategies that we have used include the following.

- Do a small study to be used as a proof of concept. Even an empirical study done using a "toy" program, might convince a project to agree to be a subject of a study if it allows practitioners to see what benefits they may conceivably get from the technology. The more relevant and important the benefits appear to be from this small study, the more likely it is to get a project interested in being a subject of a follow-on study. However, as researchers, we have to be aware that participating in an empirical study involves many risks for a production project. Realistically, it may provide few, if any, benefits to the project that is being studied. It is only later projects that will benefit in these cases. Researchers must be careful not to make promises that they will not be able to deliver. The good will of practitioners for future studies is necessary and so it is essential that claims be kept realistic.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 125 – 127, 2007.

- Volunteer to give talks particularly aimed at practitioners. One way to get potential project recruits is to have researchers describe the proposed technology to practitioners and to explicitly outline the goals of the study. If preliminary results are available, they should be presented to provide as much detail as possible. Again be careful not to make grandiose claims when they cannot be substantiated, and remember that many practitioners are wary of researchers because they have seen very few benefits of past research.
- Promise not to intrude. It is important to be aware that projects that do agree to participate in case studies are taking risks and that their time is very valuable because they generally have very strict time constraints when preliminary artifacts or new releases have to be ready. By promising to be as unobtrusive as possible, to minimize the amount of resources we use, and to ask as few questions as possible, the project is more likely to feel that you understand and appreciate their time pressures.
- Make it clear to the project how much their willingness to participate in a study is appreciated. Make it clear that you recognize that they are helping you (rather than coming in with the attitude that you have all the answers and that they should consider themselves lucky to have your attention and great ideas focused on them).
- Offer to provide extra help if necessary to make up for their help, and deliver on that offer if asked. For example, we have been asked to help out with testing when time was running short and to compute metrics for them. This was done willingly and graciously and we believe that they appreciated this and therefore are likely to be helpful again in the future. Remember, personnel on this project have friends on other projects and will themselves be personnel on other later projects that you may want as subjects. Therefore goodwill generated on this project is likely to help you win other candidate projects for later studies.

Once initial subjects for empirical studies have been found and results have been generated, recruiting additional projects to be subjects is generally somewhat easier since they can see that your technology does scale and produces promising results (assuming that's the case). One case study is almost certainly not enough to get a production project to adapt your new technology, and so additional empirical studies will generally be necessary.

Now is the time to begin in earnest trying to present your results to practitioners. Volunteer for any appropriate venue. Do not expect to be paid for telling them about your research, or even as a consultant if you convince a project to be an empirical study subject. Remember they are helping you at least as much as you are likely to be helping them. Once your research is “proven” because it has been used in several large industrial empirical studies, if the technology is deemed useful, your tool, technology, or you are all likely to be marketable and in demand. If you or it are not in demand, then perhaps the technology is not as useful as you believe, at least in its current form, or you have not yet made enough people aware of it.

3 Transferring the Technology

The empirical studies that have been performed are often the best way to win customers for your technology. If you can demonstrate that you have applied your

technology to several large production projects, having different characteristics, perhaps you can find a project that is just starting up and willing (or eager) to actually use your technology. In some cases, empirical studies that have been performed will have been retrospective rather than prospective. We have found it most convincing to have done at least one prospective study before trying to actually transfer the technology. In that way candidate for the technology can see it in use, and speak with active project personnel about their experiences.

In order to get widespread acceptance of new technology, it is often necessary to build a high-quality tool that automates the technology. If it requires one or more Ph.D. researchers to apply the technology, it probably is not ready for transfer, nor is it likely to be adopted by practitioners. It is sometimes possible to get a project who is interested in using the technology to make it into a product or production-grade tool. It may also be possible to license the technology to a professional tool-building organization. This automation should facilitate the transfer of technology.

Once the technology has been automated, it is important to continue to both publish written descriptions of the technology in appropriate venues, and also to make presentations to practitioners to show them your successful results in the empirical studies. Being able to demonstrate how the technology works on real projects is still the best way to get the technology transferred.

Relationships and Responsibilities of Software Experimentation

Giovanni Cantone

The Empirical software engineering (ESE) research community is twenty years old but still growing mainly on the internal side rather than in terms of external relationships. Let us reason on the most important of those ESE relationships and consequent responsibilities. We sketch on a static view of the problem domain entities, and their relationships, in case stereotyped relationships.

ESE Research Groups aim to improve Organizations, a term that generalizes on Developers, Producers, and Service-agencies. Organizations enact their own Processes, use Methods, utilize Tools, and employ People. Associations of (types of) Organizations do produce Standards, and (Production | Development | Service) Guidelines, including Good Production | Development Practices. The main goal is to filter products, hence organizations, which are allowed for a certain market.

Some Products, Methods, Tools are for use or consume by End–customers or for use in the processes enacted by Agencies, which provide services to organizations and citizens, who generally are not aware of the details concerning the short and long term impacts of that technology on themselves and the environment. Hence, the need and role of Authorities, which enact Ethical Control, and emit Guidelines; these include Verification guidelines and Good Laboratory Practices, (GLP)s. Because the labs of an Organization live a context that applies standards and good practices, they usually also apply (GLP)s. Vice versa, concerning other types of labs, e.g. academic labs, the application of those practices probably vary from context to context.

Software organizations are development organizations. They develop technology for end-customers and all types of organizations. Hence, there are three relevant types of software technology transfer: from software developer to software developer, or producer, or service provider, respectively. Concerning the latter, for instance, software technology has been proposed for the continual learning of physicians and paramedics. ESE should hence investigate all those types of technology transfer.

The ESE research has dependence relationships with Ethical control, (GLP)s, and related Guidelines. In other fields there are many types of experiments, to conduct orderly (in case of success only!) E0 acts on things, “in-vitro”; experimenters are the only humans allowed to come in contact; ESE scientists should simulate their experiment processes before involving volunteers. E1 acts on primary animals; seems not applicable for ESE experiments. E2 acts on formally informed human volunteers performing as subjects. $E_{n>2}$ act on dependent humans: the lower is n the greater is the level of dependency of the involved subjects. Based on the ESE practices, to the best or our knowledge, ESE experiments usually skip the stages E0, and still limit to enact (part of) E2, which is a quite frustrating fact that we should work to remove.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 128, 2007.

The (Practical) Importance of SE Experiments

Tore Dybå

Much experimental SE research involves testing a hypothesis regarding a relationship or difference between two variables. Typically, a null hypothesis (H_0) of a zero correlation or no difference between the means of the two populations is posited. The standard way of reporting results from such statistical hypothesis testing is by presenting *p*-values or information about the rejection or acceptance of H_0.

However, in an applied discipline such as SE, it is not enough to find out that one method or technique is better than another – we need evidence of *how much* better. Such evidence can be expressed in terms of an *effect size*. So, whereas *p*-values reveal whether a finding is *statistically* significant, effect size indices are measures of *practical* importance. Interpreting such effect sizes is critical, because it is possible for a finding to be statistically significant but not meaningful, and *vice versa*.

In an ongoing systematic review at the Simula Research Laboratory [2] we found that more than two thirds of SE experiments did not report any effect size measure. This lack of effect size reporting can lead to serious inferential problems and effectively reduces the practical utility of experimental results. For those experiments that reported effect sizes, or included enough descriptive statistics for effect size indices to be calculated, the median effect size was $d = 0.6$. At the same time a quantitative assessment of statistical power revealed that the experiments, on average, only had a one-thirds chance of detecting phenomena with such medium effect sizes [1]. These results indicate that SE experiments currently are both underpowered and underreported.

To further advance the field of empirical software engineering we must not only address relevant topics, we must also plan for acceptable power and report the results of our studies in a manner that could be put to use. What we regard as practically important effect sizes vary depending on the goal of the research and the fields within which its results are applied. However, in order to seriously discuss these issues and inform judgement about practical importance, effect size, or sufficient descriptive statistics for relevant effect size indices to be calculated, must be reported as part of our experimental results.

References

[1] Tore Dybå, Vigdis By Kampenes, and Dag Sjøberg. A Systematic Review of Statistical Power in Software Engineering Experiments. *Information and Software Technology*, vol. 48, no. 8, August 2006, pp. 745-755.

[2] Vigdis By Kampenes, Tore Dybå, Jo Hannay, and Dag Sjøberg. A Systematic Review of Effect Size in Software Engineering Experiments. Simula Research Laboratory, work in progress.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 129, 2007.

How to Improve the Use of Controlled Experiments as a Means for Early Technology Transfer

Andreas Jedlitschka

As stated by Robert Glass in Communications of the ACM [1], there seems to exist a gap between the information needed by industrial people and the information provided by researchers. We have, on the one hand, investigated the information need of industrial decision makers and, on the other hand, surveyed a set of controlled experiments with regard to their entropy (defined as a measure for relevant information content). The studies among industrial decision makers confirm the need for information regarding SE technology's impact on overall project cost, quality and schedule, whereas the survey of controlled experiments indicates that this information is hardly available. Accepting the importance of controlled experiments for our discipline, we conclude that in order to improve their wide-spread acceptance (usage and usability) by industrial decision makers, it is important to provide at least minimum information with regard to their need.

In terms of currently available reports of controlled experiments, we observed that the aforementioned problem is not the only one. A further problem has been identified, which hinders the widespread acceptance of the study results: It is difficult to locate relevant information. Therefore, we would like to enforce the use of reporting guidelines, which are expected to support a systematic, standardized presentation of empirical research, thus improving reporting in order to support readers in (1) finding the information they are looking for, (2) understanding how an experiment is conducted, and (3) assessing the validity of its results. We argue [2] that especially for the first issue, currently available guidelines do not provide the level of support that would be necessary for improving the use of controlled experiments as a means for early technology transfer. Whereas publishing reports of controlled experiments for the research community seems to be straightforward, the situation is different for publishing results for practitioners. They certainly care about the context, the overall results, and transfer issues, but do they care about the design and statistical analysis? Therefore, we foresee that there will be no one-size-fits-all guideline, but rather different guidelines for different user groups and purposes, e.g., for researchers aiming at aggregation or for practitioners aiming at decision support.

References

[1] Glass, R.L.: Matching Methodology to Problem Domain; In Column Practical Programmer in Communications of the ACM /Vol. 47, No. 5, May 2004, pp. 19-21

[2] Jedlitschka, A.; Pfahl, D.; Reporting Guidelines for Controlled Experiments in Software Engineering; ; In Proc. of ACM/IEEE Intern. Symposium on Software Engineering 2005 (ISESE2005), Noosa Heads, Australia, Nov 2005, IEEE CS, 2005, pp. 95-104

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 130, 2007.

Extending Empirical Studies to Cover More Realistic Industrial Development and Project Management Issues

Marek Leszak

"Real development" issues, observable in many medium size to large size software systems need to be tackled more intensively by empirical research, to increase applicability and concrete transfer of scientific results into practice. This is illustrated by three areas, covering typical problems in today's software project management.

Many predictive and prescriptive models rely on the assumption of steady-state project dynamics, at least for main driving factors. But industrial software development is facing an increasing amount of global development, offshore outsourcing, high turnover rate, requirements volatility, constrained resources (both in interval and in cost), mental project pressure, unforeseen project events, etc..

Granularity of empirical studies is often on overall *system level*. At least for large-scale and complex systems this is often not adequate. If the data would be collected and analyzed on subsystem level (product) or team level (process/project) some interesting insights can be revealed [1,2], not available if the system is studied from a 'black-box' perspective. Characteristics of different subsystems tend often not to be homogenous, causally due to e.g. different team size and expertise, and architectural dependencies. E.g. one of our case studies on defect and file distribution [2] provided evidence that a) per subsystem, delivered defects are a good estimator of in-process defects, and 2) per subsystem, the number of product requirements is highly correlated with in-process defects. In this case study such significance could neither be observed on system-level nor on module-level.

Another often neglected aspect of current empirical research is their scope w.r.t. the lifecycle phases studied. If extended towards overall development lifecycle the research results would widen their scope, leading to intensified and value-added collaboration with industrial software development. E.g. most research results on software reviews and inspections focus on code artifacts, although in real development projects typically a much higher amount of effort is spent on reviewing requirements, design documents, and test plans. It is not surprising to see evidence for review effectivity & efficiency being highly dependent (also) on type of artifact [3].

These exemplified levels of detail, in turn, requires on the industrial organization a high degree of data quality, typically not found in lower maturity organizations. Including more realistic factors from large software systems and real-life projects into empirical studies would improve model adequacy and their predictive power.

References

1. O. Laitenberger, M. Leszak, D. Stoll, K. El-Emam: Quantitative Modeling of Software Reviews in an Industrial Setting. 6th IEEE Int. Symp. on Software Metrics. Nov. 1999
2. M. Leszak, D.E. Perry and D. Stoll: Classification and Evaluation of Defects in a Project Retrospective. Journal of Systems and Software, 61/2002, p. 173-187
3. M. Leszak: Software Defect Analysis of a Multi-Release Telecommunications System.. 6th Int. Conf. on product-focussed software process improvement (PROFES). Oulu, Finland, June 2005

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 131, 2007.

Empirical Case Studies in Industry: Some Thoughts

Nachiappan Nagappan

This position paper deals with empirical case studies from an industry perspective. Empirical Software Engineering at Microsoft spans several areas including traditional software engineering, program analysis, and HCI. From the perspective of transferring ideas, tools and techniques into the product teams Microsoft Research more or less behaves like an academic department, being closer to the problem. Some of the factors to conduct empirical studies from an industry view point to get a conversation started are described below. In general empirical studies are costly in industry due to the working environment constraints and the risks associated with failure.

Cost: Cost is an important factor in most industrial empirical studies. Commercial software developers like to get an initial estimate of the cost of deploying a new software system or process. This helps them make cost benefit decisions and perform risk analysis if they are willing to adopt the new techniques.

Scale: Scale is a very important factor than needs to be understood and acknowledged in most industrial case studies. Techniques that work well with smaller systems, groups of people might not work with large systems. Understanding the reasons for such failures early on can help avoid costly mistakes.

Logistics: Conducting empirical studies in an industry environment involve a considerable amount of organizational logistics. Very often it requires becoming part of the development organization to understand trends, data, tools and techniques in order to collect in-process, important contextual information.

Tools: Most data collection, not requiring any human input has to be automated to minimize intrusion into developer's normal work. For example, tools to collect code coverage information, measure code churn etc. Such tools need to scale for large systems and should work in reasonable time with a high degree of accuracy to be adopted by product teams.

Privacy: Most product teams expect a high degree of privacy while doing empirical studies. They do not wish to have any personally identifiable information mapped backed to them with respect to their development practices.

The above factors are by no means complete or comprehensive and are just to serve as a starting point for discussion in this area.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 132, 2007.

Software Process Simulation Frameworks in Support of Packaging and Transferring Empirical Evidence

Dietmar Pfahl

Empirical research is essential for developing a theory of software development, and – subsequently – for transforming the art of software development into engineering. In particular the latter point deals with providing evidence on the efficiency and effectiveness of tools and techniques in different application contexts. An application context is defined, firstly, by organizational aspects such as process organization, resource allocation, developer team size and quality, management rules, etc., and secondly, by the set of all other tools and techniques applied in a development project.

Controlled experiments and case studies are expensive in terms of effort and time consumption. Support for making decisions on which experiments and case studies to spend effort and time would be helpful. Such support could be provided by a software process simulation framework that is adaptable to various process configurations and accepts local efficiency and effectiveness values of tools and techniques as inputs. Simulation can generate estimates for the impact on overall project performance.

Inspired by the idea of frameworks in software development, adaptable software process simulation systems can be constructed from reusable components. The scope of a reusable component comprises construction, verification, and validation activities, and associated input/output products and resources. In order to capture the main dimensions of project performance, i.e., project duration (time), project effort (cost), and end product quality, each component has three dimensions which represent recurring patterns in software process models: functional view (activities), resource view (people, techniques, tools), and quality view (products).

Besides their cost-effectiveness due to reusability, software process simulation frameworks yield the following benefits:

- Focused improvement of techniques and tools, and associated conduct of controlled experiments and case studies, thus accelerating the generation of interesting new empirical evidence about the efficiency and effectiveness of development techniques and tools.
- Standardized representation and packaging of empirical evidence about local effectiveness and efficiency of techniques and tools in varying contexts, facilitating the systematic exploration of the impact on (global) project performance at low cost.
- Improved knowledge transfer, education, and training through visualization of the impact of local effects on global performance.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 133, 2007.

Structuring Families of Industrial Case Studies

Laurie Williams

Practitioners are most influenced by results of research conducted in industrial settings. Evidence of the efficacy of a software development practice or process is best obtained through a triangulation of research findings obtained through a variety of empirical studies in various contexts. The use of an evaluation framework can enable a family of related industrial case studies in different contexts to be meta-analyzed and/or combined. Such an evaluation framework could consists of templates for specific quantitative measures to collect with associated instructions on what to include/exclude for consistent measurement collection as well as protocols for surveys and/or interviews. Groups of researchers interested in the same research question(s) can customize and evolve an evaluation framework for the technology under study.

We have developed and evolved such a framework for the study of Extreme Programming, the Extreme Programming Evaluation Framework (XP-EF), which is available as a North Carolina State University Technical Report. XP-EF is currently at Version 1.4. The composition of Version 1.0 was initially developed from relevant literature. We refined the framework in several research cycles with industrial partners on four multi-release industrial case studies. We have also revised the process based upon input from anonymous reviews of our publications from researchers in the community, through discussions with members of the International Software Engineering Research Network (ISERN), and through presentation of our work to researchers and practitioners. Through these cycles of use and external feedback, the research and practitioner communities have provided input on the guidelines and artifacts embodied in the framework. We have conducted four case studies guided by the XP-EF. The reported results of these case studies reference the evaluation framework used, enabling precise replication of metrics collection and data analysis.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 134, 2007.

Empirical Software Engineering: Teaching Methods and Conducting Studies

Claes Wohlin

Abstract. Empirical software engineering has grown in importance in the software engineering research community over the last 20 years. This means that it has become very important to also include empirical studies systematically into the curricula in computer science and software engineering. This chapter presents several aspects and challenges to have in mind when doing this. The chapter presents three different educational levels to have in mind when introducing empirical software engineering into the curricula. An introduction into the curricula also means increased possibilities to run empirical studies in student settings. Some challenges in relation to this is presented and the need to balance educational and research objectives is stressed.

1 Introduction

Empirical software engineering has established itself as a research area within software engineering during the last two decades. 20 years ago Basili et al. [Basili86] published a methodological paper on experimentation in software engineering. Some empirical studies were published at this time, but the number of studies was limited and very few discussed how to perform empirical studies within software engineering. Since then the use of empirical work in software engineering has grown considerably, although much work still remains. A journal devoted to empirical software engineering was launched in 1996 and conferences focusing on the topic have also been started. Today, empirical evaluations and validations are often expected in research papers. However, the progress in research must reflect on education. Computer scientists and software engineers ought to be able to run empirical studies and understand how empirical studies can be used as a tool for evaluation, validation, prediction and so forth within the area.

This chapter presents some views on education in empirical software engineering. In particular, the chapter contributes by identifying three different educational levels for empirical software engineering. Furthermore, the chapter stresses the possibility to run both experiments and case studies as part of courses and how this facilitates research. However, to perform research as part of courses, with the educational goals of a course in mind, requires a delicate balance between different objectives. The research goals must be carefully balanced with the educational goals. Different aspects to have in mind when balancing the different objectives are presented. Finally, the need to develop guidelines for running empirical studies in a student setting is stressed. Guidelines are needed both to ensure a proper balance between different objectives and to, within the given constraints, get the maximum value from empirical studies run in a student setting. It is too simplistic to disregard empirical studies with

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 135 – 142, 2007.

students just because they are students instead there is a need to increase our understanding of how empirical studies with students should be conducted and to what extent results from them could be generalized.

The chapter is outlined as follows. Section 2 provides an overview of some related work. In Section 3, three different educational levels to consider when introducing empirical studies into the curricula are presented, and different types of empirical studies to use in relation to courses are discussed. Section 4 presents some aspects to have in mind when balancing educational goals and research goals. Finally, a brief summary and conclusions are provided in Section 5.

2 Related Work

The literature on education in empirical software engineering is limited. The literature primarily describes either specific experience from a course, such as [Höst02], or from performing a research study with students as subjects [Thelin03]. Most empirical studies conducted in an educational setting are controlled experiments, although some articles are more related to studies of student projects and hence could be classified as case studies [Höst02, Berander04]. Some exceptions exist, where authors discuss the use of students in empirical studies for different purposes [Carver03].

Anyway, there is a need to improve the way we both teach and conduct empirical studies in software engineering. However, there are many challenges. Software engineering is primarily concerned with large scale software development and hence, for example, the use of controlled experiments in laboratory setting is not straightforward. This is true both when it comes to student learning and to conducting research in an academic setting. We have to understand and define how to conduct controlled experiments in laboratories to make them useful in a larger context. This makes the challenges quite different from other disciplines using experimentation. There is a lot to learn from other disciplines, but there is also a need to address the specific challenges when conducting experiments in software engineering.

When experimentation was introduced into software engineering, the main focus was on running experiments. As the work progressed, more focus has been put on the actual methods used. The two books on experimentation [Wohlin99, Juristo01] are good examples. Researchers have also started addressing how results from different empirical studies should be combined either through meta-analysis [Hayes99, Miller99] or using a more evidence-based approach [Kitchenham04]. Other researchers have addressed the challenge of building families of experiments [Basili99] and hence plan for combining experiments rather than trying to combine existing studies. This is also closely related to the issue of replication [Shull02]. Some experiences from conducting realistic experiments are presented in [Sjøberg02], in particular the article argues for funding running experiments and hiring professionals to participate in them. This is one possible way. However, we believe that we have to better understand how to transfer results from laboratory experiments with students as a complement to running larger experiments and hence also much more expensive experiments with professionals. In particular, it is a challenge to both teach empirical methods in an effective way and to simultaneously run studies that are valuable from a research perspective or from a technology transfer perspective.

When it comes to actually trying to understand the role of students as subjects in controlled experiments in software engineering, the first article addressing the topic explicitly was published in 2000 [Höst00]. The article presents an empirical study where both students and professionals participate, and then the article compares the two subject groups. In this case, no differences could be identified in performance. In another study, it is concluded that it is better to experiment within project courses than as separate exercises [Berander04]. Based on the conviction, that it is too simplistic to look solely at students vs. professionals, a scheme has been developed to include both experience and motivation [Höst05]. In another study, an interesting observation was made when a situation was found where the results from experiments with students became possible to generalize. It was a situation when the students knew one of the methods used much better but still the other method came out as being the best to use [Staron06]. When it comes to work on objects to use in experiments, the research is very limited. It is mentioned in passing in books, but no research actually addresses the challenges that have been identified.

In addition to the above, a large number of actual experiments have been published. A survey is presented in [Sjøberg05]. Furthermore, two books with a collection of empirical studies, including controlled experiments, have also been published [Juristo03, Wang03].

3 Teaching Empirical Methods

This section addresses issues to consider when teaching empirical software engineering on different educational levels, including bachelor, master and PhD level. Furthermore, the section discusses the use of different empirical methods in particular experiments and case studies.

3.1 Educational Levels

Empirical software engineering may be taught in at least three distinct ways:

- Integration in software engineering courses
  One way of making empirical studies a natural part of evaluating and assessing different methods, techniques and tools is to integrate empirical work as assignments or studies within other courses. This means that in, for example, a course on software design different ways of performing a design can be compared and evaluated empirically. Another example is to include an experiment in a verification and validation course, where, for example, unit testing and inspection could be compared. In particular, it may be interesting to evaluate which type of faults that are found with the different techniques. Integration into courses early in the curricula means that students get exposed to and used to apply empirical methods.
- Separate course
  It is also possible to run a separate course on empirical software engineering. The major advantage is that the course becomes very focused on empiricism and hence it becomes easier to provide assignments that include designing studies, reviewing empirical work and also possible write an empirical paper. The latter may be done without actually running a study, i.e. the focus could be in identifying, designing

and discussing an empirical study. A potential drawback may be that it is hard to get a separate course on empirical software engineering into the curricula. It is often many different topics that should be covered in a software engineering program.

- Inclusion in research methodology course
  The need for a research methodology course normally becomes evident when students are approaching their thesis work, independent of whether it is a bachelor, master or PhD thesis. Thus, it may be natural to include an empirical software engineering component in a more general research methodology course. Other components may be the use of an analytical approach or the use of simulation.

The above three alternative ways of teaching empirical software engineering may be mapped to educational levels. The mapping is based on experiences at Blekinge Institute of Technology to integrate empirical software engineering into a general software engineering program. At Blekinge, students may get five different degrees in software engineering. The first three are regarded as being on undergraduate and graduate level and they are (counting years from the start at the university): 1) university diploma in software engineering after two years, 2) a bachelor degree after three years and 3) a master degree after 4.5 years. The two latter includes a thesis, where the thesis on the master level normally is more research-oriented than the thesis on bachelor level. On a research level, two different degrees are awarded: 1) licentiate degree after two years of research studies and 2) a doctoral degree after four years of research studies. The licentiate degree includes a year of course work and a thesis equivalent to one year of full time research. The doctoral degree includes 1.5 years of course work and a thesis equivalent to 2.5 years of research. The work to licentiate level is expected to be included in the doctoral degree. To simplify a little and make the Swedish system comparable to an international context, the main focus is set on the bachelor, master and PhD levels. These three levels are also the ones included in a joint agreement within the European community to ensure compatibility between countries within the community.

In our experience, it is suitable to try to integrate assignments with some empirical parts into courses on the bachelor level. It is also possible to successfully perform experimental studies with students as subjects on the bachelor level. However, it is crucial to ensure a very clear educational goal with such studies. This is particularly important on the bachelor level since the students are still quite far from research.

On the master level, it is still important to integrate empirical methods and studies into other courses. The students are now approaching the research level and in particular they are expected to write theses with a research component, and hence it is possible to run experimental studies that are more research focused. However, it is still very important to discuss the outcome of the studies and relate to whatever topic they are studying. Before the master thesis, it is suitable to run a course on either empirical software engineering or a more general research methodology course with an empirical component. The actual choice is dependent on whether the thesis work is expected to be only empirical or if other research approaches are also possible. It also matters whether the objective is to run a joint course between several programs, for example, a joint course between software engineering and computer science. The latter is the situation at Blekinge and hence methods for empirical studies are included in a more general research methodology course.

Finally, on the PhD level, we have experience from teaching specific courses on controlled experiment in software engineering, empirical software engineering and also a course on statistical methods for software engineers. The latter course is based on having data from a number of empirical studies in software engineering. The course is given using a problem-based approach. This means that the students are given a set of assignments connected to the data sets provided. No lectures are given on how to solve the problems and no literature on statistics are provided. It is up to the students to find suitable literature and hence statistical methods to use based on the assignments and the data available to them. The students practice statistical methods useful both for case study data and data from controlled experiments. Students provide individual solutions that are peer reviewed before a discussion session. In the discussion session, differences in solutions are discussed and the teacher shares her or his experience with the students. The sessions include discussions on both the statistical methods used and the interpretation of the analysis in a software engineering context. This course is run for the second time in 2006 with participants from five different Swedish universities.

3.2 Different Types of Studies

Most empirical studies run in a student context are controlled experiments. This poses some challenges. First of all, it is important to ensure that any study run contribute to the learning process of the students. Controlled experiments are often run with small artifacts and they are run standalone, i.e. they are not normally integrated into a development project. Given that the students in most cases will work in software projects, it is crucial to show how the knowledge gained from an experiment may be important also in the larger development context. Second, it is important that the main objective is on the educational goals. Controlled experiments may be important either from a research perspective or from a technology transfer perspective, but they may not be the most important study for the researcher if using students as subjects. A discussion on these challenges when conducting controlled experiment with students is provided in Section 4.

Experiments are most common when involving students. However, case studies should not be forgotten. They could often provide valuable information in a more project-like context. In many software engineering educations, development projects are an important part of the curricula. This does not only include individual projects. In many cases, smaller (4-6 students) projects or more large scale (12-18 students) software projects are run. These projects are not industrial projects, but they have a number of things in common with industrial projects, such as requiring that people contribute to a common goal. In some cases, these types of projects may have industrial customers. An important research question would be to identify in which cases student projects would be relevant to use in research. Once again, the research focus should not interfere with the educational goals. In [Berander04], an example is provided where it was shown that students working in projects behaved more as in industry than students participating in a controlled experiment.

4 Balancing Objectives

A particular challenge is to balance different objectives when conducting empirical studies as part of the curricula. If a study is part of a course, then the research objectives should not be allowed to dominate over the educational goals. In other instances, it is not only a matter of research; it is also a matter of using an empirical study in an academic context as a stepping stone in a technology transfer process. In other words, a researcher would like to evaluate a method, technique or tool in an academic setting before moving it out to, for example, an industrial research partner. The balance between objectives puts some constraints on the actual empirical study from a research perspective:

- Clearly connected to educational goals.
- Use of students is always challenged.
- Mandatory participation may affect the results, but optional participation is not necessarily better.
- Objects in the study must be reasonable for the different perspectives.
- Comparisons between competing methods, techniques or tools must be fair.

The latter two items may require some further elaboration. Students as subjects have been discussed more in the literature than the use reasonable objects, for example, in [Höst00, Carrver03]. From an educational perspective objects in a study should not be too large due to time constraints for the students. Moreover, it may be preferable if the objects contain certain aspects or constructs that relate to what have been taught in a course. This may include, for example, certain constructs in a design method. From a research perspective, it may be preferred if the objects resemble industrial use. This may mean that objects ought to be larger, and it may also mean that certain parts of, for example, a design method should not be used or should be used. This may particularly be the case when the research is conducted as part of a technology transfer process to a specific company in which case it would be preferable to resemble that actual intended use at that specific company. This may be contradictory to the educational objectives. This requires both a delicate balance between objectives and an increased understanding of how objects should be constructed to help in balancing the objectives.

The other issue related to the objects is the fairness. It is not fair to teach one method and briefly introduce another method, and then compare them. This would clearly favor the method having been taught. Thus, it is important that methods, techniques and tools are introduced in similar ways to ensure comparability from a research perspective. There is one potentially valuable exception. If students have a thorough introduction to one out of two competing, for example, test methods and a brief introduction to the second method, and the second method comes out as the best then it is interesting. This is interesting since the opposite would normally be expected and hence it seems like the second method is not only better; it is probably superior.

In summary, too much focus is set on the use of students in research studies. There is a need to better understand the whole context of an empirical study in a student setting. The challenge is to develop our research methodology so that we better know

how to gain the most knowledge from empirical studies with students. To simply disregard such studies is too simplistic and it may mean that learning opportunities are lost. As realistic studies as possible is good for both education and research. Thus, we must work with improving our understanding of how to run and how to make use of empirical studies within education.

5 Summary and Conclusions

This chapter presents some ways to integrate empirical software engineering into the curricula. Some issues related to this introduction are highlighted. In particular, the need to balance educational objectives with research objectives is stressed. It is argued that both education and research would benefit from as realistic empirical studies as possible when performing studies involving students. Thus, it can be concluded that the challenge is to integrate empirical software engineering and empirical studies into the curriculum and maintain research relevance and quality. Instead of being negative towards studies with students, we should increase our understanding of how empirical studies with students can be used as part of our research process. We must address questions such as:

- How is empirical software engineering best introduced into the curricula to ensure that students are both able to run empirical studies and capable of understanding their value?
- Is it possible to effectively combine educational and research objectives when performing empirical studies in a student setting?
- Can empirical studies with students be a natural stepping stone in technology transfer?

Empirical software engineering has established itself as an important area within software engineering. However, it still remains to effectively introduce it into computer science and software engineering curricula, and to address several challenges in relation to the introduction.

References

[Basili86] V. R. Basili , R. W. Selby , D. H. Hutchens, Experimentation in software engineering, IEEE Trans. on Software Engineering, 12(7):733-743, 1986.

[Basili99] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. IEEE Trans.on Software Engineering, 25(4):456–473, 1999.

[Berander04] P. Berander. Using students as subjects in requirements prioritization. In Proc. 3rd Int. Symposium on Empirical Software Engineering, pages 167–176, 2004.

[Carver03] J. Carver. L. Jaccheri, S. Morasca and F. Shull. Issues in using students in empirical studies in software engineering education. In Proc. Int. Software Metrics Symposium, pages 239-249, 2003.

[Hayes99] W. Hayes. Research synthesis in software engineering: A case for meta-analysis. In Proc. 6th Int. Software Metrics Symposium, pages 143–151, 1999.

[Höst00] M. Höst, B. Regnell, and C. Wohlin. Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering: An International Journal, 5(3):201–214, 2000.

[Höst02] M. Höst. Introducing empirical software engineering methods in education. In Proc. Int. Conf. on Software Engineering Education and Training, pages 170-179, 2002.

[Höst05] M. Höst, C. Wohlin and T. Thelin. Experimental context classification: Incentives and experience of subjects. In Proc. Int. Conference on Software Engineering, pages 470-478, 2005.

[Juristo01] N. Juristo and A. M. Moreno. Basics of Software Engineering Experimentation. Kluwer Academic Publishers, 2001.

[Juristo03] N. Juristo and A. Moreno (editors). Lecture Notes on Empirical Software Engineering, World Scientific Publishers, 2003.

[Kitchenham04] B. A. Kitchenham, T. Dybå, and M. Jørgensen. Evidence-based software engineering. In Proc. Int. Conference on Software Engineering, pages 273–281, 2004.

[Miller99] J. Miller. Can results from software engineering experiments be safely combined? In Proc. 6th Int. Software Metrics Symposium, pages 152–158, 1999.

[Shull02] F. Shull, V. R. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonca, and S. Fabbri. Replicating software engineering experiments: Addressing the tacit knowledge problem. In Proc. 1st Int. Symposium on Empirical Software Engineering, pages 7–16, 2002.

[Sjøberg02] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In Proc. 1st Int. Symposium on Empirical Software Engineering, pages 17–26, 2002.

[Sjøberg05] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N-K Liborg and A. C. Rekdal. A survey of controlled experiment in software engineering. IEEE Trans. on Software Engineering, 31(9): 733-753, 2005.

[Staron06] M. Staron, L. Kuzniarz and C. Wohlin. Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments. Journal of Systems and Software, 79(5):727-742, 2006**.**

[Thelin03] T. Thelin, P. Runeson and C. Wohlin, "An Experimental Comparison of Usage-Based and Checklist-Based Reading", IEEE Transactions on Software Engineering, 29(8):687-704, 2003.

[Wang03] A. I. Wang and R. Conradi (editors). Lecture Notes in Computer Science: Empirical Methods and Studies in Software Engineering: Experiences from ESERNET, Springer Verlag, 2003.

[Wohlin99] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. Experimentation in Software Engineering – An Introduction. Kluwer Academic Publishers, 1999.

Educational Objectives for Empirical Methods

Natalia Juristo

Abstract. Empirical Methods (EM) cover a broad field and not all software professionals need to be equally acquainted with this area. A BSc student of Computing, an MSc student or a PhD student will put EM to different uses in their professional life. Therefore, we should train them differently. To decide what to teach to whom we have analyzed what are the EM educational objectives based on the target behavior for different job profiles. In this paper, we discuss each type of student and the reasons for the target EM behaviors to be achieved through the training.

Keywords: Empirical Software Engineering, Education, Experimentation.

1 Introduction

Today most software engineers extol the virtues of a particular technology or technique based on their opinions. In SE subjective opinions are more prevalent than objective data. For software development to really be an engineering discipline and be able to predictably build quality software, practitioners must transition from developing software based on speculation to software development based on facts.

If we want future developers to lay aside perceptions and build on data, we have to focus on training today's students. We need practitioners trained in working with facts rather than assumptions. We need practitioners that understand the importance of demanding proof of a particular technology's superiority before jumping into its use. We want practitioners that know how to measure and monitor improvements and changes that occur during their developments.

For future practitioners to reach this level of maturity they need to be trained in EM [1]. Unfortunately, there is no agreement on what they should be taught about this subject. A main difficulty is that EM is a very large area encompassing many topics. At the lowest level, the field is confined to understanding the value of data. At the most advanced end of the spectrum, it covers topics like the accumulation of evidence through aggregation of results from different empirical studies (ES). To further complicate this issue, different types of students (undergraduate, MSc, PhD…) need different EM training.

The content of this paper is based on the experience of teaching EM to students at undergraduate, master and PhD levels and our observations of students' reactions to their exposure to different EM topics.

To decide what to teach each type of student we have studied how we expect a person who is now a student to behave towards EM in his or her future professional life. In other words, we have analyzed EM educational objectives based on the target behavior for different job profiles.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 143 – 150, 2007.

The paper is organized as follows. Section 2 identifies possible EM student types. Section 3 presents educational objectives for each identified student type. The other sections focus on each type of student, the reasons for the target EM behavior to be achieved through the training, and how to reach such educational objective. Section 4 deals with undergraduate students, section 5 discusses MSc students, section 6 focuses on SE PhD students and section 7 discusses the training for future experimental researchers. Finally, section 8 presents the conclusions.

2 Use of Empirical Methods in Different Job Profiles

EM covers a broad field, and not all software professionals need to be equally acquainted with this area. A BSc student of Computing, an MSc student or a PhD student will put EM to different uses in their professional life. Therefore, their EM training should be different too.

To establish the contents of the EM training for different student types, we have defined the use to which four different job profiles put EM:

- **Developers** (analysts, designers, testers, etc.) have to be aware that not all software technologies are universally applicable and some are better than others depending on the context. They should learn that objective data are helpful for finding out more about reality, while subjective opinions can lead to mistaken perceptions of reality. Developers should be aware that engineering disciplines are grounded on objectivity.

  This job profile fits BSc in Computing or SE graduates.

- **Project managers** should appreciate the value of objective data for making grounded decisions. Managers should know how to collect data from software projects and how to analyze it. Understanding publications that report on ES can help them to better evaluate technologies. Finally, software managers may find it useful to perform case studies to test technologies in their organization.

  We match this job profile to holders of an MSc in SE.

- **SE researchers** should be able to understand the results of ES, as this will be of benefit to their own research. This knowledge can stimulate a deeper and more regular collaboration with Empirical SE (ESE) researchers. For instance, providing relevant hypotheses to be checked in ES, working with ESE researchers on interpreting the results, etc.

  This job profile fits holders of a PhD in SE.

- For **Empirical SE researchers** EM are the object and material of their work. They perform ES to generate the SE empirical body of knowledge. Therefore they need to be EM experts.

  We match this job profile to holders of a PhD in SE specialized in ESE.

3 Target Behaviors for Empirical Methods

The goal of the teaching/learning process is to transform student behavior as a result of educational situations. Educational objectives are useful to describe the learning

Table 1. Bloom's taxonomy of objectives

DOMAIN	CATEGORY	MEANING
Cognitive	Knowledge	Acquisition of knowledge
	Comprehension	Grasp the meaning of information
	Application	Use of learned information to solve problems
	Analysis	Breaking down of a whole into its component parts
	Synthesis	Applying items or parts to form a new whole
	Evaluation	Judging something based on particular values
Affective	Reception	Awareness of the existence of certain phenomena
	Response	Active participation
	Valuing	Internalization of a set of values
	Organization	Form an internally consistent value system
	Characterization	Creation of a particular lifestyle from values

transformations to be achieved in the student. In other words, they are a statement specifying the expected student learning outcomes.

To understand the appropriate training on EM for the different student types we have worked on educational objectives. We use Bloom's taxonomy [2], shown in Table 1, as a means of more systematically formulating the learning objectives.

Bloom's classification makes a distinction between two domains within learning. The cognitive domain encompasses knowledge-related learning, whereas the affective domain contains the goals that describe attitudes, interests and values.

The affective domain objectives are often left out of instruction on other computing subjects. But they need to be considered in EM training since it is just as important to produce a change in future software engineers' attitudes and values with respect to empiricism as it is to convey knowledge. Therefore, it is necessary to work at the affective, not only at the cognitive level.

Table 2 lists the educational objectives for EM training for the different student types that we have identified. Notice that the different types of student are conceived as rungs further up the educational ladder. Therefore, the educational objectives at a one level assume that the student already satisfies the educational objectives of lower levels.

In the following sections we detail the educational objectives for each student type and how a course can achieve these goals.

4 EM for Undergraduate Students

For undergraduate students there are educational objectives in both the affective and cognitive domains. Affective domain objectives are critical for this type of students since the main goal here is to produce a change in their outlook. This change in today's students should produce a shift in tomorrow's software professionals' attitudes and values towards EM.

We should teach students that opinions and beliefs are not always true. To achieve this goal it is not enough to just tell them. The best way to change future software

Table 2. Educational objectives for EM in SE

LEVEL	DOMAIN	CATEGORY	MEANING
Undergraduate	Affective	Reception	Be aware that opinions and beliefs can be wrong
		Response	Evaluate reality based on objective data
		Valuing	Accept the importance of collecting and using data
	Cognitive	Knowledge	Acquire measurement and data analysis terminology
		Comprehension	Interpret measurements
		Application	Use data analysis techniques
Master	Affective	Reception	Be aware that own perceptions can be wrong
		Response	Participate in ES as subjects and compare perceptions with collected data
		Valuing	Accept the importance of data for decision making
		Organization	Accept the role of data collection and ES in a software organization
	Cognitive	Knowledge	Acquire basic notions of EM
		Comprehension	Interpret ES results to understand findings
		Application	Perform case studies
PhD in SE	Affective	Characterization	Behave in line with the philosophy of EM
	Cognitive	Knowledge	Acquire advanced knowledge about EM
		Comprehension	Interpret ES and relate them to theoretical research
		Application	Conduct ES
		Analysis	Identify the appropriate ES to be run under certain circumstances
PhD in ESE	Cognitive	Synthesis	Plan and design ES
		Evaluation	Evaluate ES design, quality, results, etc.

engineers' attitude towards EM is for them to actually experience the fact. A way to do this is through exercises and games. Play at analyzing situations where they can form a predictive opinion and show them how data contradict their beliefs.

We set our students the following game. Imagine you are a project manager. Some projects have to be done within a very short time. You want to choose the more efficient of two programmers, Mary or Susan, to work on a project. Mary arrives at work early and stays till late. Susan usually arrives later than Mary and leaves before her. Susan spends less time at work than Mary. If you need the task to be done fast, who will you choose? We encourage students to discuss the situation and think whether they need more information to make a decision. Students tend to select Mary for the project.

We then provide students with productivity data. This data set shows that Mary is more productive than Susan. We now discuss metrics with students. We make them understand that productivity rather than working time was the metric they needed to ground their decision. We want them to understand that data also have their risks.

Finally, we analyze the productivity data sets using data analysis techniques. The result is that there is no significant difference between Mary's and Susan's productivity. We discuss at this point the utility of data analysis techniques and how very often just looking at the data is not enough.

Apart from this exercise, we play other similar games with the students to work on the affective domain and make them value objectivity, data, metrics, analysis techniques, etc.

The main cognitive content of the EM course is basic concepts of EM and data analysis techniques. In theory, if all faculty subscribed to an EM philosophy, a separate course would not be necessary, as the affective and cognitive objectives could be distributed across several degree subjects. Unfortunately, this is not the case at all computing schools. Until ESE has permeated the SE community, a course that covers these teaching objectives is necessary.

In our program the students have knowledge of both statistics and SE when they choose this EM elective. However, there are possible variations on this scenario. For example, students may have no knowledge of statistics, SE or either. Table 3, shows alternatives for these situations. If students have no knowledge of statistics, the objectives related to data analysis would have to be removed. Instead of teaching by examples that the students themselves analyze, they would be given examples that have already been analyzed. Another option is for the EM course to cover the basics of statistics. If the students have no knowledge of SE, the examples used would have to be substituted for examples with which the students are familiar (using examples and data taken from everyday life, for instance.).

Table 3. Variations on the EM course for undergraduate students

	No knowledge of SE	**Basic knowledge of SE**
No knowledge of Statistics	– Without data analysis – Examples from outside SE	– Without data analysis – Examples from SE
Basic knowledge of Statistics	– All objectives – Examples from outside SE	– All objectives – Examples from SE

5 EM for MSc Students

Again the educational objectives for this student type build upon the affective domain. We go one step further in modifying students' attitudes and values. We aim to show students that even their own perceptions can be wrong. Besides we want future software managers to make a commitment to incorporate data collection and ES in their organizations.

At this level, students will benefit from participating as subjects in an experiment. The goal is twofold. On the one hand, they will form perceptions of the techniques they have been using during the experiment. Their perceptions about these techniques will now be based on first hand use rather than opinions or beliefs. They tend to believe that their own experience is more reliable than opinions. We discuss their perceptions in the classroom: which technique of those used during the experiment seems to be more efficient, for instance. Afterwards we show them the results of analyzing the data. Very often their perceptions do not match the objective results yielded by analyzing the data collected during the experiment. This exercise lays stress on the importance of collecting data to support decisions, rather than using manager perceptions.

The second goal of participating in an experiment is to get experience on how to perform ES. The students do not just participate as blind subjects in the experiment, we teach them experimental design, analysis, etc. So they get some insight into organizing ES.

The cognitive course content is basic EM concepts, data collection, types of ES, performance of case studies, and data analysis techniques. This content trains future software managers in the collection of data during software development, the execution of case studies, and the interpretation of publications of ES to assess their utility and be able to use the results.

Note that participating in an experiment as subjects implies that they have the knowledge to apply the techniques used during the experiment. There are different ways of implementing this. One way is to do the ES outside the EM course. In this case, the experiment would be part of the course that teaches the topic exercised in the experiment. If the ES is run as part of a different course, there are two possibilities:

- The two courses are taught one after the other. In this case, the support course (on which the experiment is to be conducted) should be taught before the EM course. The discussion of the students' perceptions during the ES, the analysis of the data, etc., will be part of the EM course.
- The two subjects are taught simultaneously. In this case, they should be scheduled to assure that the experiment has already been conducted by the time the discussion on perceptions starts and data analysis techniques are taught.

If the experiment is run outside the EM course, there are additional benefits for the support course. It has been found [3] that participating as subjects in an ES during a SE course is well received by the students compared to other more conventional ways of teaching/learning (lectures, book, exam, literature study, etc.). Other benefits of student participation in ES are [4]: students' attention is kept at a good level, student evaluation is not based en episodes, students are encouraged to develop a critical mind.

6 EM for PhD Students

As Table 2 shows, we make a distinction between SE PhD students and what we have called ESE PhD students. The main difference is the research area of these PhD students. The latter are researching in ESE, and their thesis is an empirical one. This does not necessarily mean they are taking different PhD programs.

We want future SE researchers to be acquainted with the need to validate research and the existence of methods to experimentally validate SE research. We do not expect to make an EM expert of every SE researcher. Our aim is for SE researchers to at least check the feasibility of the solutions they propose. It would be asking too much to expect them to perform ES to validate their ideas. In other fields (like physics, for instance) there are theoreticians and experimentalists. We see ESE researchers as the experimentalists of SE. ESE PhD students are the future ESE researchers.

Although we do not expect SE PhD students to become experts in ESE, they should be capable of designing an experiment for their thesis. Also they should know how to analyze data, as well as to interpret whether or not an ES is sound and the implications of its results. This will give SE PhD students a good enough understanding of the techniques involved in ESE to be able to work side by side with experimenters. On the one hand, this implies being able to understand the ES performed, as well as the results they yield, which they might use to direct their own theoretical research. On the other hand, it entails taking part in running ES.

The contents of a course on EM for SE PhD students include: an introduction to ESE (terminology, types of ES, etc.), design and analysis of experiments.

The teaching methodology proposed to achieve this goal is again eminently practical. We want the students to do exercises applying the theory. Therefore, we give them a case (completed experiment) that they use to apply the theory. The students apply each stage of experimentation to this case. We ask the students to prepare a design, and discuss with them the different designs proposed, including ours. Then we provide the data we collected and ask the students to analyze it. We show and discuss the results of our analysis.

The course contains another practical part. We ask the students to plan and design an experiment to test the ideas they propose in their thesis. This does not mean we expect the experiments to be performed. This depends on how difficult it is to get subjects for the ES, the type of thesis, and many other details. But the exercise of thinking about how this experiment should be run gives the future SE researcher some practice in ES that can help to close the gap between ESE and SE.

It is important for what we have called ESE PhD students to expand their knowledge of more advanced areas of ESE, like, for instance, qualitative techniques, experiment replication, aggregation of ES results, etc. This knowledge expansion can be achieved by students attending courses on different ESE subjects. One possible scenario is a joint doctorate among several universities with ESE researchers. The experts on advanced ESE topics could teach courses at different universities or students could visit other universities to attend seminars.

7 Conclusions

The goal of the teaching/learning process is to transform student behavior as a result of educational situations. Educational objectives are a statement specifying the expected student learning outcomes. In this paper we have analyzed what to teach to different types of students considering how a software practitioner is expected to behave with respect to EM in her profession.

To do this analysis we have divided software professionals into three groups: developers, project managers and SE researchers. Here we propose a number of EM educational objectives and how they can be achieved within a course. The discussion has been organized around three different type of students: undergraduate, master and PhD.

If you would like to learn more about EM courses that are being taught visit http://openseminar.org/ese

References

[1] Natalia Juristo and Ana M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer. 2001

[2] Lorin W. Anderson, David R. Krathwohl, Benjamin Samuel Bloom. *Taxonomy for Learning, Teaching, and Assessing, A: A Revision of Bloom's Taxonomy of Educational Objectives*. 2nd Edition. Longman Publishing Group. 2000

[3] Martin Höst. Introducing Empirical SE Methods in Education. *Fifteenth Conference on SE Education and Training (CSEET)*. 2002

[4] Jeffrey Carver, Letizia Jaccheri, Sandro Morasca, Forrest Shull. Issues in Using Students in Empirical Studies in SE Education. *Ninth International Software Metrics Symposium (METRICS)* 2003

On "Landscaping" and Influence of Empirical Studies

Frank Houdek

Experimentation is considered to be an important element in technology transfer. Empirical results on new approaches should help to persuade decision makers to apply these approaches in their environment. But to what degree are decisions for new approaches driven by empirical results at all? From a subjective point of view, decisions for new approaches seem often to neglect empirical results at all. What are the reasons for this? To analyze this situation, we first have to see that there are at least three dimensions that affect technology selection:

1. *Point in time for selection of new approaches.* Usually, industry is not waiting for new approaches to be suggested. Established processes are maintained and sometimes improved evolutionarily. The window of opportunity, i.e. the period of time when a company seeks for new approaches is usually comparably small. Often, this window opens in conjunction with improvement initiatives (e.g. a CMMI assessment and a follow up program to cure identified deficits). Pushing approaches outside that window might only help to increase awareness – technology adoption is bound to these windows.

   As a consequence, proposals for new approaches should be (1) available during that time window and (2) related to typical deficits identified during such assessments (to be provocative: no one cares about optimal reading techniques it itself, but a efficient set of technologies to improve the KPA quality management might be very welcome).
2. *Subjective degree of relevance.* Technology adoption is driven by two main factors: need for change and trust that proposed technology might help (i.e. the risk of the particular technology is comparably low).
3. *Relevant empirical results.* Let us assume that the amount of relevant empirical knowledge in software engineering is depicted as a world map, we would see that only limited areas have already been discovered yet. Many areas are still completely unknown. Unfortunately, there is even no such map yet and building such a map should be a major activity in the next year.

   However, a number of criteria to identify promising trails that should be followed (i.e. questions that should be taken into account) can be already identified. Three examples are given below:
   - Cost-Benefit trade-off analysis (not primarily looking for best technique, but for the cost-efficient one)
   - Robustness: Likeliness that technique survives even under project pressure
   - Required background: Can the technique be beneficial be used with, e.g. electrical or mechanical engineers?

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 151, 2007.

Involving Industry Professionals in Empirical Studies with Students

Letizia Jaccheri and Sandro Morasca

Empirical studies are often carried out with students because they are viewed as inexpensive subjects for pilot studies. Though the literature has mostly focused on their external validity, we believe that there are a number of other issues that need to be investigated in empirical studies with students. In our past research, we have identified four viewpoints, each of which needs to be taken into account when carrying out successful empirical studies with students: researchers, teachers, students, and industry professionals. Each viewpoint can be seen as a stakeholder of an empirical study with students, with specific and possibly conflicting goals. The stakeholders also have risks from participating in empirical studies, which need to be identified and minimized.

At any rate, the final goal of carrying out empirical studies with students is carrying out empirical studies in industrial organizations and establishing collaborations with them. It is therefore useful to involve industry professionals in empirical studies with students, and they should actually play all of the stakeholders' roles.

- Professionals as students. This is the case of industrial training or continuous education, and it may be the case of the participation of industrial professionals in university software engineering classes. This could help establish a strong communication channel between academia and industry by showing empirical software engineering may provide value added to them.
- Professionals as customers. Professionals can play the role of the customers for the empirical studies with students in software engineering classes. This may not entail any direct or deep involvement with the empirical study itself. Showing interesting results may help establish a good collaboration.
- Professionals as researchers. Empirical investigations may be at least partially designed and run in the context of software engineering courses by industrial professionals who are interested in the research results. Here, the degree of involvement is certainly higher than in the previous case, as is the interest in cooperating with academia.
- Professionals as teachers. Professionals may be invited by academic institutions to share their expertise with students, for a few lessons in a class or teaching entire classes. In this context, professionals may be willing to carry out studies with students and even be the main driving force behind them. Being both teachers and researchers, professionals will have to find an optimal trade-off between the conflicting goals of either role. Carrying out empirical studies with students is a learning experience for professionals (as well as for professors). The experience that professionals and teachers gather by running empirical studies with college students can be used when running empirical studies with professionals in industrial settings.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, p. 152, 2007.

Industry-Research Collaboration
Working Group Results

Lutz Prechelt and Laurie Williams

During this working group, we gathered a set of guidelines for establishing and maintaining fruitful industry-research collaborations. We divided these guidelines into the collaboration phase (awareness, contact, setup, and research, as will be defined) and, within each phase, to whom the guideline applied (researchers, practitioners, or both). The agreed-upon deliverable from this session was for a subset of the group to submit a longer version of this report to IEEE Software or the Empirical Software Engineering journal as a set of guidelines.

1 The Awareness Phase

During the awareness phase, both the research and practitioner organizations learn about the typical constraints, expectations, and interests of the other side and reflect on their consequences for collaboration and on the opportunities that can arise.

Research organization:

- Be aware that the practitioner organization will be concerned that the research has a return on investment for the software process improvement (SPI).
- Understand that business comes first. The practitioner organization cannot go out of their way to accommodate research. The degree to which this applies depends on the current pressure in the organization.
- Understand that the research must fit with the timelines of the practitioner organization. Be ready when the window of opportunity opens.
- Understand that finding a perfect project without limitations is unlikely. Trade scientific weaknesses for industrial realism in these studies.

Industrial organization:

- A collaboration with a research organization is especially well-suited for strategic improvement activities. Maintaining and evolving technology-based unique selling points in the long run usually requires such collaborations.
- Do not expect a research organization to solve all of the most urgent problems, rather, expect important contributions to the solution.
- Understand that researchers may be deeply interested in issues you deem to be uninteresting. Work together to determine if this disconnect of interest can be reconciled or if it is because the researcher is interested in a state-of-the-art problem which is perhaps too theoretical in current form or if the researcher is interested in a research topic that is unlikely to have practical value.
- Understand that research organizations are not able to be flexible are with respect to manpower (because both the researcher's time and the supply of graduate students may be limited) and scheduling (which may depend on semester and graduation rhythms).

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 153 – 157, 2007.

- Collaborating with a research organization may provide you with an inside track to capable graduates-to-be.
- Published results of studies conducted with your organization can be helpful for marketing and public relations.
- Research funding is often necessary. The academic does not have the time for detailed analysis, and student research assistants usually expect to be paid.

Both:

- Realize that one beauty of the collaboration is the sharing of perspective. Respect your differences in priorities and background and what the other accomplishes in their own field, and learn from each other.

2 The Contact Phase

During the contact phase, both organizations identify one another as suitable collaboration partners, meet representatives of the other side, and build initial trust.

Research organization:

- To meet potential practitioner collaboration partners, visit practitioner conferences, interest group meetings, and fairs. Learn about the problems they consider important. Give understandable presentations that relate to these problems. Talk about relevant related work in addition to your own.
- Contact your former graduates to see if they are possible research partners.
- Have practitioner-oriented information on your web pages, posted prominently. Usually this does not replace active partner acquisition but it can substantiate a first contact that was established via other channels.
- Prepare a one-minute elevator pitch: Who are you? What research questions are you interested in? Why?
- Understand the current business goals that drive a specific practitioner's interest in a collaboration. Discriminate between their urgent and important problems and merely interesting ones. Working on the latter will not get you solid, continuous support.
- If the practitioner organization may be right but the relationship with your first contact person does not work out, ask that person to help identify a contact in that organization that is more appropriate, open, supportive, and influential.

Industrial organization:

- To find potential research collaboration partners, get to know the research groups of nearby universities. Visit them. Invite them to visit your company. Challenge their research approach and see whether they are open to collaboration.
- Advertise specific collaboration interests at suitable scientific meetings.
- Perform a reality check of the research organization: The researchers cannot know the details of your situation, but they should understand the basic realities, constraints, and forces of practical software development.
- When you propose research to a research organization, try to understand how attractive that research is scientifically and how it could be made more so.

Both:

- Collaborations in which both parties are local are generally more effective. Frequent, direct contact may alleviate communication problems and subsequent conflicts. However, geographically-separated collaborations can work as well.
- In discussing research proposals, work together on the question "What's the benefit for the practitioner organization?"

3 The Setup Phase

In this phase, representatives of both organizations identify and agree on a specific study that they want to pursue together and both explicitly formulate their respective goals and hence increase mutual trust. Further stakeholders meet, buy-in to the research, and get to know each other.

Research organization:

- Understand the current software process at the practitioner organization.
- Plan for making the *economic* payoff of your research highly plausible; perhaps extend the study setup accordingly.
- Unless you perform action research, carefully minimize the invasiveness of your study design. Explain to the practitioners where and how you reduced invasion, where you could not, and why. Find alternative routes together.
- When you present the research goals and approach to practitioner stakeholders, tailor the presentation to their various roles, viewpoints, language, and level of expertise. For each stakeholder group, clearly answer why they might be interested in the research or at least its results.
- Make it clear that scientific publication is for the researcher what production is for the practitioner, and that scientific publication requires detail. A publication strategy describing what will be published, how the data will be sanitized and what will be confidential should be defined early.
- Consider accepting unattractive research content if attractive opportunities appear to be feasible in the future, but only if you are competent at pursuing the former.
- Participants may be concerned if they feel that the study can be used for individual monitoring purposes. Ensuring that this is not the case may improve the cooperation from them.

Industrial organization:

- Identify a champion who will drive the study with the development team.
- Specify how you will reserve the resources required for continuously supporting the research and describe the method of escalation, if needed.
- Be ready to accept publication of information coming from within your organization. Make sure you clarify early what you cannot accept at all to be published and where sanitizing (e.g. by normalization or anonymisation) will be required.

- Obtain broad support for the research, so that the study can continue despite organizational change or personnel turnover.
- Determine if any practitioners want to become co-authors of the resulting publications. These practitioners should be active participants in both the research and the writing (as prescribed by the Vancouver Convention[1] article).

Both:

- Consider running a pilot study first to assess risks.
- Avoid having more than one intermediary on each side. Stakeholders that actively participate in the research should be allowed to communicate as directly as possible. Research is just like software development in this respect. Direct communication is usually cheaper and always much more reliable.
- Intellectual Property issues need to be discussed and settled, particularly if it is expected that a patent-able asset will be created. The research team may need to sign a non-disclosure agreement.
- Make sure there is sufficient management support on both sides.
- Formulate your expectations for the collaboration: Research opportunities, active support from within the organization, permission to publish.
- Likewise, formulate your minimum requirements for a barely acceptable collaboration. This may weaken your position for negotiations, but is helpful for building trust and is essential for long-term collaboration.

4 The Research/Deployment Phases

The partners execute the study, work together, build further trust, obtain results, and deploy new technology into other projects.

Research organization:

- Get to know as many stakeholders of the practitioner software process as you can. Build personal relationships. They are often the source of crucial insights and the basis for keeping up the research until a successful conclusion.
- Have a plan for making faster progress during the practitioner organizations' low-pressure periods, when the practitioners can offer better support.
- If your research will modify the process of the practitioner organization, make sure you provide initial training and continuing support.
- Treat the practitioner organization like a customer: Speak with one voice, be solution-oriented, gently work around idiosyncrasies.
- Steer clear of company politics.
- During the study, provide timely and accurate feedback, not just of preliminary results, but also of other observations that may be of value for the practitioner organization.
- Demonstrate both the short- and long-term payoff from your research.

[1] *Authorship: rules, rights, responsibilities and recommendation.* http://www.jpgmonline.com/article.asp?issn=0022-3859;year=2000;volume=46;issue=3;spage=205;epage=10;aulast=Sahu

- Plan to sanitize the results obtained from the research prior to publication such that proprietary information about the company cannot be obtained or even extrapolated from the publication. Allow adequate time (at least a week or two) for company lawyers to review publications prior to their submission.

Industrial organization:

- Make sure the researchers receive adequate support from your staff.
- Students graduate, so expect researcher turnover.
- Treat the researchers like consultants: Assume they know more about their field of specialty but need to be informed about the peculiarities of your situation.
- During the study, provide timely and accurate feedback, regarding both the content of the research and your perception of the collaboration.
- Deploy the practices that can be considered successful research into additional development projects.
- Deploy the practices that can be considered successful research into additional development projects.

Both:

- Regularly review and revise your common research plan. Communicate problems and complaints openly. Perform risk management (analysis, planning, monitoring, mitigation, follow-up).
- For long-term collaboration, always perform a short postmortem on each study for mutual feedback and optimization of the future research process.

Acknowledgements

The member of this group were: Vic Basili, Jeffrey Carver, Tore Dybå, Tracy Hall, Frank Houdek, Andreas Jedlitschka, Marek Leszak, Michael S. Mahoney, James Miller, Jürgen Münch, Nachiappan Nagappan, Markku Oivo, Lutz Prechelt, Carolyn Seaman, Rick Selby, Dag Sjøberg, Helen Sharp, Larry Votta, Laurie Williams, and Claes Wohlin.

Teaching Empirical Methods to Undergraduate Students *Working Group Results*

Austen Rainer, Marcus Ciolkowski, Dietmar Pfahl, Barbara Kitchenham, Sandro Morasca, Matthias M. Müller, Guilherme H.Travassos, and Sira Vegas

1 Introduction

In this paper, we report the experiences of a working group who met, as part of the 2006 Dagstuhl Seminar on Empirical Software Engineering, to discuss the teaching of empirical methods to undergraduate students. The nature of the discussion meant that the group also indirectly considered teaching empirical methods to postgraduate students, mainly as a contrast to understand what is appropriate to teach at undergraduate and postgraduate level. The paper first provides a summary of the respective experiences of the participants in the working group. This summary is then used to informally assess the progress that has been made since the previous Dagstuhl Seminar, held in 1992. The paper then reviews some issues that arise when teaching methods to undergraduate students. Finally, some recommendations for the future development of courses are provided.

2 Progress Since the 1992 Dagstuhl Seminar

Table 1 and Table 2 provide a summary of the working group participants' experiences of teaching empirical methods to software engineering students on under-graduate and post-graduate degree programmes at Universities around the world. The table indicates a wide range of experiences. Of particular interest is the age of the course. Table 1 indicates that all of the courses started sometime after the 1992 Dagstuhl Seminar. This suggests that the empirical software engineering community has made progress in establishing 'mechanisms' for preparing the next generation of researchers and industry practitioners in using and interpreting the output from empirical methods.

The two tables indicate that some clear differences between the courses offered on the different degrees, for example:

- Between those courses that concentrate on detailed issues (e.g. statistical analysis, measurement, experimental design) and those courses that concentrate on more general issues (e.g. project management, software technology evaluation).
- Between those courses that emphasise the development of a research focus (e.g. investigate a research question), and those courses that emphasise a practitioner focus (e.g. an investigation to aid practical decision-making).
- Between those courses that primarily focus on empirical methods, and those courses that primarily focus on some aspect of software engineering (e.g. software design, software quality) but incorporate material on empirical methods.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 158 – 162, 2007.

Table 1. Summary of empirical methods courses offered on degree programmes

University (Country) *Degree* / course title	**Duration**	**First started**
Keele University (UK)		
*MSc IT** / Professional practice	2 semesters	2002
MSc IT and Management / Professional practice	2 semesters	2002
MSc / Project	16 weeks	2003
BSc CS and Information Systems / SE project management	1 semester	2000
BSc CS / SE	1 semester	2005
Universität Karlsruhe (Germany)		
4th year of Diploma in Informatics / Empirical SE	1 semester	2002
University of Calgary (Canada)		
3rd year of BSc SE / Software Metrics	1 semester	2002
University of Hertfordshire (UK)		
3rd year of BSc / Empirical evaluation in SE (elective)	2 semesters	2005
MSc / Research methods (required)	2 semesters	2002
Federal University of Rio de Janeiro/COPPE (Brazil)		
DSc and Msc Systems Engineering and CS / Experimental SE	1 quarter	2001
DSc and Msc Systems Engineering and CS/ Advanced SE	1 quarter	2004
Universidad Politécnica de Madrid (Spain)		
5th year of BSc Computing / Advanced SE (elective)	1 semester	2002
PhD Computing / Experimentation techniques in SE (elective)	1 semester	2002

*IT: Information Technology; SE: Software Engineering; CS: Computer Science

Although there are clear differences in the structure and content of the courses, there was general agreement about the need to encourage students to become reflective practitioners (cf. [1]). Participants differed, however, in what content they considered was required to help encourage that reflection. For example, some participants considered that a statistical understanding is important for reflective practice, whilst others believed that a broader understanding of the general nature of research and evidence would be important for reflective practice.

3 Some Issues

Various issues were identified by the participants:

- Most students do not yet have experience of industrial software projects. Consequently, practical activities that encourage students to reflect (using empirical methods) on their own, personal software engineering may be more effective than activities that encourage students to reflect on software projects. Similarly, it may be more effective to teach empirical methods as part of a course that focuses on some aspect of a software project e.g. using empirical methods to evaluate software quality.

- An undergraduate course on empirical software engineering research may be very different in content to many other courses on BSc Computer Science and Software Engineering degrees (e.g. programming, computer systems and networks, formal systems, database) therefore undergraduate students may struggle with the unusual content of a course on empirical software engineering research.
- Students are unlikely to have experience of reading and understanding academic journals and conference papers. The more practitioner-oriented journals (e.g. *IEEE Software*) and trade magazines (e.g. *Computer Weekly*) can provide suitable material for some courses.

Table 2. An indication of the content of some of the empirical methods courses

University *course title:* Topics
Keele University *SE project management*: Evidence based SE, including systematic review *SE*: Evidence based SE, including systematic review
Universität Karlsruhe *Empirical SE*: Empirical research methods; methodology of controlled experiments; Data representation, analysis and interpretation; statistical methods
University of Calgary *Software metrics*: Measurement theory; empirical research (e.g. types of studies, general guidelines); GQM
University of Hertfordshire *Empirical evaluation in SE (elective):* Evidence based SE; argumentation; high-level reflection on evidence; practical decision making
Federal University of Rio de Janeiro/COPPE *Experimental SE*: experimental research methods; methodology of experimental studies; Data representation, analysis and interpretation, statistical methods. *Advanced SE*: Evidence Based SE including systematic review; Meta-Analysis.
Universidad Politécnica de Madrid *Advanced SE (elective)*: Introduction to measurement; introduction to experimentation; data analysis techniques

4 Where Next? Some Recommendations

The participants considered how we can help lecturers teach empirical methods to undergraduate students, and what we might learn from other disciplines in the teaching of empirical methods.

Two broad approaches were considered to help lecturers teach empirical methods to undergraduate students: the sharing of resources through a website, and the development and use of textbooks that emphasis empirical evidence.

A website was considered to be a potentially useful mechanism for sharing resources because it would overcome intranet obstacles, and would allow originators of materials to decide what to share. For the site to be successful originators of material would need to maintain their own material over time. Examples of resources to share include:

- Materials e.g. copies of lecture slides, tutorial exercises, small practical experiments.
- Articles, or references to articles e.g. the course on Empirical Evaluation in Software Engineering at the University of Hertfordshire uses the Standish Group's 1994 CHAOS Report, and Jørgensen and Moløkken's [2] and Glass's [3] critiques of that CHAOS report as a basis for discussion.
- Guidelines and standards e.g. a number of courses (some listed in Table 2; see also [4]) use the guidelines of Evidence Based Software Engineering. Other courses will use guidelines for conducting experiments, reporting statistical results etc.
- Common terminology. The empirical software engineering research community does not seem to have settled on a standard (common) terminology. For example, the term 'experiment' has a range of different meanings within the community, from referring to a randomised, controlled trial to a 'pseudo-experiment' to a pilot study to any kind of empirical study. Consequently, there is a need for the research community to settle on a standard terminology, and for this terminology to be communicated to the next generation of researchers and research-aware software practitioners.

A range of textbooks that include evidence was also identified, and distinctions where made between those textbooks that provide a review of the empirical work conducted in an area (e.g. [5]), those textbooks that concentrate on empirical research methods (e.g. [6-9]), those textbooks that use evidence to support the statement of laws and theories (e.g. [10]) and those textbooks that use empirical evidence to support recommendations being made about how to conduct software engineering (e.g. [10, 11]). The later category of textbook (i.e. those that use evidence to support recommendations) seems to be the most lacking, and the most needed by *undergraduate students*.

The participants briefly considered how other disciplines may contribute to the teaching of empirical methods to undergraduate students on computer science and software engineering degree programmes. Four broad examples were suggested:

- Nursing, specifically midwifery: researchers in this field rarely have the opportunity to conduct controlled experiments and most often rely on observational field studies for their investigations.
- Education: a degree of training is often needed for an intervention to have an affect (e.g. a teacher must be trained in a certain educational technique) and so the training effects may also affect the outcome of the study.
- Human-computer interaction: there are some well-defined, 'small' experimental studies in human-computer interaction that could be used as examples of empirical studies when teaching empirical methods to undergraduate students.
- Behavioural sciences: the behavioural sciences have a long history, and therefore experience, of designing studies that take account of human subjects e.g. risks to the validity of an experiment. In other words, the behavioural sciences can make a contribution to the teaching of methodological issues.

The group also recognised, however, that one needs to be careful in borrowing from other disciplines, because technologies may not work outside of their original contexts or may need to be modified carefully in some way.

5 Conclusion

The experience of these working group participants is that the teaching of empirical methods has progressed considerably since 1992, with a number of degree programmes either now offering courses specifically on empirical methods, or including empirical methods on related courses (e.g. software quality). At the undergraduate level in particular, the focus of these modules should typically be on encouraging students to become effective reflective practitioners rather than effective researchers. Subsequent postgraduate courses can then develop research skills.

References

1. Schön, A., The Reflective Practitioner: How Professionals Think in Action. 1983: Basic Books.
2. Jørgensen, M. and K. Moløkken, How Large Are Software Cost Overruns? A Review of the 1994 CHAOS Report. 2005, Simula Research Laboratory. p. 8.
3. Glass, R.L., IT Failure Rates--70% or 10-15%? Software, IEEE, 2005. **22**(3): p. 112-111.
4. Jørgensen, M., T. Dyba, and B. Kitchenham. Teaching Evidence-Based Software Engineering to University Students. in 11th IEEE International Software Metrics Symposium. 2005. Como, Italy, 19-22 September 2005.
5. Fenton, N. and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach. 2 (revised printing) ed. 1997, London: PWS Publishing Company.
6. Juristo, N. and A.M. Moreno, eds. Basics of Software Engineering Experimentation. 2001, Kluwer Academic Publishers.
7. Wohlin, C., Experimentation in Software Engineering: An Introduction. International Series in Software Engineering. 1999: Kluwer Academic Publishers.
8. Schulze, R., Meta-analysis: A Comparison of Approaches. 2004: Hogrefe & Huber Publishers.
9. Christensen, L.B., Experimental Methodology. 2003: Allyn & Bacon.
10. Endres, A. and D. Rombach, Empirical Software and Systems Engineering: A Handbook of Observations, Laws and Theories. 2003: Addison Wesley.
11. Juristo, N. and A.M. Moreno, eds. Lecture Notes on Empirical Software Engineering. Series on Software Engineering & Knowledge Engineering. 2003, World Scientific Publishing.

Technology Transfer and Education
Discussion and Summary

Andreas Jedlitschka, Dietmar Pfahl, and Kurt Schneider

Abstract. This is a short summary of the talks and discussions during the Technology Transfer and Education session. The main theses of the keynote presentations are summarized and a short summary of remarkable discussion points is presented.

1 Overview

The session started with four keynote presentations, two on technology transfer and two on education. Although it is not useful to repeat those contributions in detail, their basic messages are summarized and complemented with observations from the group work and the plenary discussions.

2 Keynotes on Technology Transfer

2.1 Elaine Weyuker: Mutual Learning

In the first keynote, Elaine Weyuker pointed out that technology transfer is a way of educating both practitioners and academics (professors and students). Empirical studies within software companies are of particular value. They will be challenged by the complexity of real-world software development. Practitioners appreciate empirical results much more if they demonstrate that an innovative software engineering concept actually works in a real environment. Typically, practitioners will only be convinced of a new technology if studies provide not only a proof of concept, but also a proof of scalability of the concept to industrial complexity.

However, proof of concept and scalability through empirical research is not sufficient to transfer a new technology into industry. Empirical studies need to be conducted in different types of projects, in order to show that a new concept works in various environments. This is also a good way to understand how a new technology might have to be adjusted to different environments in order to unfold its full potential. Empirical studies conducted in industry are an important source of learning for academia. The real environment cannot be emulated by laboratory experiments conducted with students in a university environment.

Other important issues for successful (large-scale) technology transfer include the provision of tool support for a new technology and the willingness to educate practitioners appropriately. The latter does not refer to the writing of scientific papers that are presented at academic conferences, but to giving talks to practitioners using their language and exciting them with success stories.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 163 – 167, 2007.

2.2 Larry Votta: Individuals, Not Institutions Collaborate

In the second keynote, Larry Votta stressed the point that technology transfer is based on collaboration between individuals, not institutions. Successful technology transfer can only happen when both sides, individuals from academia and industry, identify common goals and problems and develop a model of mutual education. Larry Votta sees technology transfer as a game of "give and take" for both involved parties. While academia has the resources to build up in-vitro laboratories that can experiment with new ideas and come up with prototypes and models that represent the core idea of a new technology without overloading it with all the necessities imposed by the complexities of industrial software development, practitioners can help academic researchers to check the validity of their models in real environments (in-vivo laboratories) and to adjust them to the needs of industry.

2.3 Insights from the Plenary Discussion

The plenary discussion following immediately after the keynotes on technology transfer generated insights that were shared by the group, but also raised new questions without being able to give converging answers.

- Technology transfer can only be successful if the technology is mature enough, i.e., sufficiently tested in various contexts, scalable, and supported by tools.
- (Academic) Researchers need to understand that their collaboration with industry is not about receiving money for consulting services but about the opportunity to test a new technology in a realistic environment.
- For academic researchers, it is essential to know the state-of-the-art. This supports technology transfer, because it helps to avoid the "not-invented-here" syndrome and the temptation to exclusively try to sell one's own technologies.
- Furthermore, for academic researchers it is essential to know the state-of-the-practice, because that helps them to speak the language of the practitioners.
- The complete life-cycle of technology development and transfer is important, i.e., combining initial feasibility studies (often in vitro) with studies in industrial environments (in vivo).
- In order to assist technology scouts from industry in finding new and interesting research results, it is essential that academia keeps their web pages related to technology development up-to-date and informative.

There was also a number of issues that could not be resolved during the discussion:

- The need of academia to frequently publish new research results nurtures the tendency to reduce the complexity of research problems. How can the aspiration of academia towards simplification and generalization be brought into congruence with the needs of industry to find customized solutions for complex products and processes?

- Due to the lack of opportunity and time, academic researchers tend to lack hands-on-experience with real industrial software development. How can this threat to successful technology transfer be mitigated?
- It is often difficult to make the first step: Young researchers and new research groups need to establish those relationships described by Votta and Weyucker. When empirical research results are presented to practitioners who did not yet have a chance to work with empirical software engineering researchers, they also need a path to a good relationship.

3 Keynotes on Education

3.1 Natalia Juristo: Teaching Empirical Study Skills

In her keynote, Natalia Juristo reported her experience with teaching empirical software engineering to different groups of students, i.e., undergraduates, graduate students (Master level), graduate students (PhD level), and PhD students with a research focus on empirical software engineering. She pointed out that the main challenge was to identify which aspects of empirical software engineering should be taught to which group of students.

Based on experience from teaching different classes at different levels at the Universidad Politécnica de Madrid, the following focus areas were identified per student group:

- *Undergraduate:*
  understand data and perform data analyses for small data sets;
- *Graduate (Master level):*
  understand empirical studies; prerequisite: involvement as experimental unit in a previous course;
- *Graduate (PhD level; general software engineering):*
  understand and practice the design and planning of empirical studies; apply knowledge to empirically validate one's own PhD research;
- *Graduate (PhD level; focused on empirical software engineering):*
  understand and practice advanced topics of empirical software engineering, e.g., qualitative methods, aggregation, and replication.

It was pointed out that graduate students pursuing a Master degree need to know empirical methods but do not necessarily have to validate their research results by means of a controlled experiment. According to Natalia Juristo, teaching empirical methods to undergraduate students is the biggest challenge, since students on that level are not made aware sufficiently that software engineering is a feedback-driven discipline. Possible ways to overcome this difficulty is to better integrate empirical software engineering with the core software engineering disciplines. In order to stress that empirical software engineering is not a discipline but an attitude, it might be helpful to change the label "empirical software engineering" to "evaluation of software engineering technologies".

3.2 Claes Wohlin: Integrated or Separate Empirical Software Engineering Courses

In the second keynote, Claes Wohlin raised the question of whether empirical software engineering should be taught as a separate course or better be integrated with canonical software engineering courses. Based on his experience, he suggested that undergraduate students and first-year graduate students participate in studies, while more advanced students should actively study empirical methods, review papers that report on empirical studies, and conduct empirical studies. This should be done in a problem-based manner. Towards the end of his keynote, Claes Wohlin raised questions related to the teaching of empirical methods to software engineering students:

- What is the best way to combine research and teaching? Can it be supported by involving students as experimental units, or would this conflict with the teaching objectives?
- What are suitable objects of empirical studies involving students? This question arises in particular for case studies. Does it make sense to do case study research with student projects? What are the requirements that these kinds of case studies need to fulfill in order to be useful?
- Can empirical studies be a starting point for technology transfer, or is there a potential conflict? Is there a danger that the teaching objectives are compromised if industry-oriented research is integrated into teaching?

3.3 Insights from the Plenary Discussion

The discussion triggered by both keynotes brought up several observations and conclusions.

- The most important being the suggestion to develop a wide spectrum of interaction between universities and industry, involving students. Some universities make sure that graduate students do their Master or PhD related work within companies, and integrate empirical studies with that. This arrangement creates a setting similar to real-world project work, with external requirements and project management, and focus on products that actually are useful for local industry. A successful example of that kind of collaboration is Barry Boehm's project repository of student projects which has been maintained for several years.
- Frank Houdek used the metaphor of a research "world map" of empirical research topics and suggested to explore blank areas rather than the crowded islands (like reviews and inspections).

4 Reports from the Working Groups

Following the keynotes and discussions one working group per topic was set up:

- Working Group 1: Industry collaborations
- Working Group 2: Experiences in teaching empirical methods

The results of both working groups were presented by their respective leaders, Laurie Williams and Austen Reiner (in this volume), and subsequently discussed.

The group on industry collaboration distinguished several phases of building a relationship between researcher and company.:

- During the awareness phase, the contact phase, and the set-up phase, it is important for researchers to consider the other party's point of view.
- During that period, psychological issues, misunderstandings, and unrealistic expectations play a big role.

Their work group summary contains a rich collection of concrete hints and recommendations for all phases of industry collaboration.

The education working group raised a couple of important issues:

- One could take advantage of general empirical methods courses without explicit connection to software engineering. This allows for providing many different examples, including non-software examples from other engineering disciplines.
- Student internships in software organizations have proven to be effective in many ways; in particular, they give students a taste of real-world problems in software development, and they help software organizations to get in touch with potential future employees or research collaborators.
- Students should be given the opportunity to conduct small experiments. When doing this, it is important to allow them to fail, and give them a chance to learn from mistakes (related to both the empirical work and the research under study – in case they are evaluating their own work results).

Empirical Software Engineering Research Roadmap
Introduction

Richard W. Selby

1 Roadmap Motivation

The gathering of leading Empirical Software Engineering researchers at Dagstuhl provides a unique opportunity to capture the current challenges facing the field. Our gathering enables deep discussions that identify critical issues, discuss promising opportunities, and outline future directions. A typical framework for organizing ideas and plans from thought leaders is the definition of a roadmap for a field, and the researchers gathered at Dagstuhl have agreed to define a roadmap for Empirical Software Engineering research.

The following sections describe what we mean by a roadmap and elaborate on the roadmap process, uses, benefits, types, and structure. We then introduce an example skeletal roadmap for Empirical Software Engineering that serves as a starting point for Dagstuhl working group discussions. In the subsequent sections, we document the working group's outcomes and describe the resulting overall roadmap in a final summary.

2 Roadmaps Overview

Robert Galvin, who is the former Chairman of the Board of Directors for Motorola, states "a 'roadmap' is an extended look at the future of a chosen field of inquiry composed from the collective knowledge and imagination of the brightest drivers of change in that field" [Gal98]. The optimal process for gathering and selecting the content of roadmaps is to include as many professionals as possible in workshop discussions. This process allows all suggestions to be considered and evaluates objectively the consensuses that emerge. The process should incorporate treatment for minority views and individual advocacies. The following steps help illustrate the process:

- Step 1: Identify goals
- Step 2: Identify key functions supporting the goals
- Step 3: Identify key technologies supporting the functions
- Step 4: Identify the contribution of Empirical Software Engineering to the technologies, functions, and goals

Roadmaps provide multifaceted uses and benefits. Roadmaps communicate visions, attract resources from business and government, stimulate investigations, and monitor progress [Gal98]. They become the inventory of future possibilities for a particular field. They facilitate interdisciplinary networking and teamed pursuit. Roadmap rationale and visual "white spaces" can conjure promising investigations.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 168 – 171, 2007.

Roadmaps typically establish directions and facilitate coordination and assessment of progress. For example, science and technology roadmaps identify or set future directions and facilitate technology assessments. Industry and government roadmaps set industry directions and coordinate execution. Corporate roadmaps set and monitor directions, coordination execution, and help manage products, platforms, and portfolios.

According to Galvin [Gal98], roadmaps can comprise:

- statements of theories and trends,
- formulation of models,
- highlighting of linkages among and within sciences,
- identification of discontinuities and knowledge voids, and
- interpretation of investigations and experiments.

Roadmaps can also include the identification of instruments, charts, and graphs needed to solve problems as well as potential showstopper challenges. A roadmap's structure commonly adopts an application domain centric viewpoint in order to:

- describe the state of the art,
- describe the state of the practice,
- identify overall key issues and social benefits and impacts,
- agree on a vision for the future,
- determine criteria for achieving progress,
- identify enabling research,
- determine enabling technologies, and
- build chaired working groups.

3 Roadmaps for Empirical Software Engineering

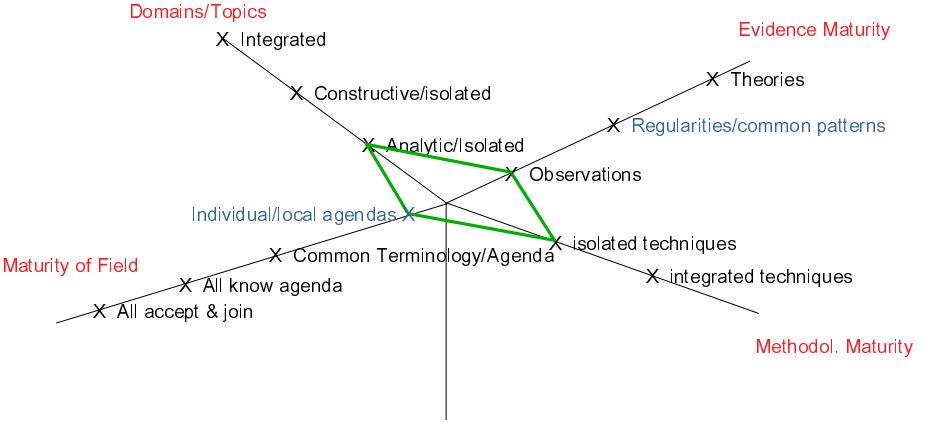
For an Empirical Software Engineering (ESE) roadmap, some important questions to ask for different application domains are as follows:

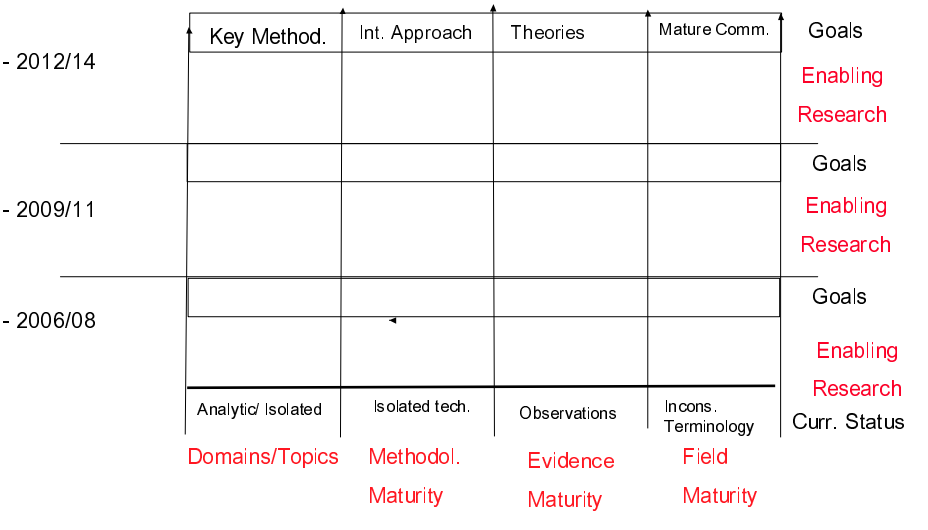
- Economical and societal driving forces
  - Why is there a need for ESE research?
- Objectives
  - What goals should be attained?
  - What solutions can ESE offer to the stated needs?
- Scientific challenges
  - What are the challenges we – as scientists – are facing?
- Technological driving forces
  - Which key technologies are expected to push development in this area?
- Bottlenecks that hinder progress
  - What hinders the development of ESE in a specific area?
  - What are the societal, economical, and technological obstacles?
- Future research activities
  - Which activities should be (financially) supported?
  - What are the coarse timeframes of development in the areas?

At Dagstuhl, our specific roadmapping approach defines the following objectives:

- Identify important organizational "dimensions" for the ESE field
- Define goals for each dimension
- Identify current status for each dimension
- Establish progress scales along each dimension from current status to goals including short-term, mid-term, and long-term

To initiate discussion, we present an example roadmap for Empirical Software Engineering:





In support of the roadmap, the above example table provides potentially additional detail.

The following sections document the working group's outcomes and describe the resulting overall roadmap in a final summary.

Reference

[Ga198] Robert Galvin, "Science Roadmaps," Science, Vol. 280, May 8, 1998, p. 803.

Roadmapping
Working Group 1 Results

Ross Jeffery

1 Introduction

The group discussion developed the chart shown in Figure 1 below. In this chart we show four dimensions:

1. Industry relevance and impact
2. Confidence of understanding
3. Cohesiveness and maturity of the research field
4. Rigor of research, methodological consensus and trust in the research.

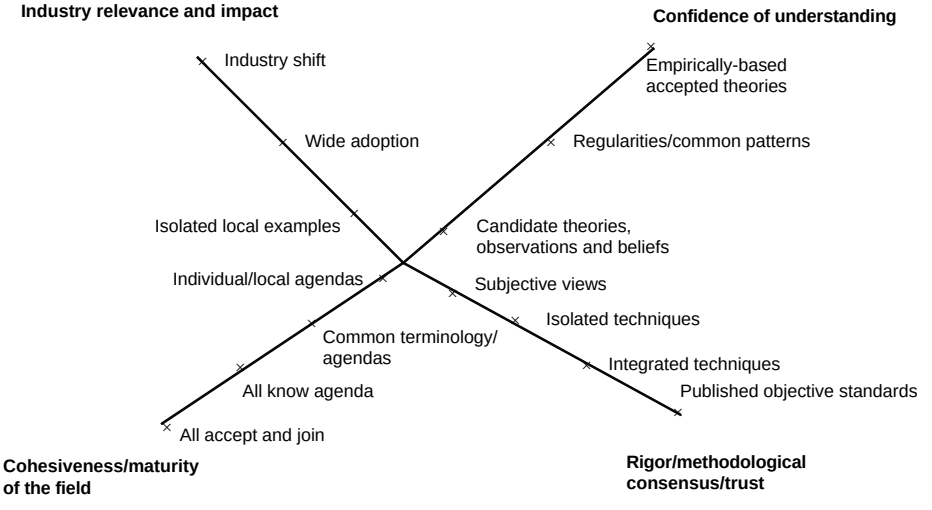


Fig. 1. Kiviat-style Chart of Empirical Software Engineering Research Roadmap

It was decided that research domain or topics of research was not one of the dimensions of this chart but rather an over-riding characteristic that applied to all instances of the chart. This means that the chart would be instantiated for each research topic or domain investigated, such as domains like embedded software, financial systems, defence systems, automotive systems or topics like testing, inspections, architecture and so on. This recognises that the current understanding and agreement might be at different stages of maturity for different domains or topics. The chart shows the standard characteristic of rigor versus relevance that has been explored in the past in research in the information systems community, and adds

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 172 – 174, 2007.

aspects of confidence of understanding and maturity of the field. It is likely that even these two dimensions are related to rigor and relevance. However the group did not explore these possible relationships.

2 Industry Relevance and Impact

This dimension was seen to be able to progress from isolated examples through wide adoption to a significant shift in industry practice.

3 Cohesiveness and Maturity of the Field

This dimension begins with isolated research agendas and proceeds to a point where the research community has all accepted the agendas and joined in the research. The mid points might be described as firstly some groups having common terminology and agendas, followed by all in the field knowing these agendas, before finally all accept.

4 Rigor, Methodological Consensus and Trust

This dimension progresses from isolated subjective views with limited or no empirical support, followed by isolated techniques which have been subject to some local rigorous investigation and justification. The next stage seen by the group was when these isolated techniques become further integrated with other groups of researchers and other ideas to form a set of accepted integrated research techniques. The final stage is the publication of accepted research method standards which are widely used by the community.

5 Confidence of Understanding

Whereas the previous dimension concerned the rigor of the investigative method, this dimension concerns the belief in the understanding generated by the investigation. Initially we might have only candidate theories, some observations and/or beliefs about a phenomenon, a relationship or a technology. At this stage we have limited evidence to support these. At the other extreme we have empirically validated and accepted theories that explain the phenomena under consideration. Between these3 two end points we might have regular patterns observed or partially validated candidate theories.

6 Time Stamping the Development of the Empirical Research Roadmap

Table 1 shows the dimensions of Figure 1 mapped into a possible time scale. This table also reveals example mechanisms proposed to achieve some of the outcomes

noted in the Kiviat chart. For example, it is proposed that an indirect measure of the field having achieved industry relevance and impact would be to have achieved a successful merged industry and academic conference on empirical software engineering by 2012. Other entries in this table indicate the type of outcome that will be needed if we are to further mature the research discipline as opposed to suggested measures of the outcome. An example of this is the suggested need for a mechanism to referee research agendas in order to have a cohesive research agenda by 2009.

Table 1. Empirical Software Engineering Timed Research Roadmap

2012/14	Successful, merged industry/ academic ESE conference	Published objective research standards	Empirically based accepted theories	Accepted and defined research agendas	Goals
2009/11		Mechanism to collect and validate techniques, proposed standards		Mechanism for refereeing agendas	Goals
2006/08	Isolated examples of joint industry academic conferences	Isolated techniques	Candidate beliefs and observations	Proposed agendas	Goals
	Industry relevance and impact	Rigor and methodological consensus	Confidence of understanding	Cohesiveness, maturity of the field	

Roadmapping
Working Group 2 Results

Marcus Ciolkowski and Lionel Briand

1 Introduction

Commonly agreed roadmaps are an indicator for mature research fields. This also implies that roadmaps are revised and updated on a regular basis. During the Dagstuhl seminar, four parallel working groups addressed this issue. This chapter summarizes the results of one discussion group. Instead of looking at the initial roadmap in its breadth, we decided to detail it in one dimension.

2 Detailing the Roadmap

We picked the Methodology Maturity to detail the roadmap. Methodology can be refined into several subdimensions, among them the mode of investigation, the type of data analysis, data collection procedures, and others, such as purpose (explorative, confirmative). The result of the working group is by no means complete or final.

2.1 ESE Methodology Maturity Refined

The **mode of investigation** concerns the type of study conducted. This can be a controlled experiment, quasi-experiment, case study, action research, a survey, or even a simulation. Thereby, controlled experiments are currently only done in academic settings; that is, not in the field (¬F), quasi-experiments have be executed in the laboratory and in the field, while all other modes of investigations have been done online, in field studies (F), with the exception of simulation, where this classification does not apply.

Data analysis: Numerous approaches for data analysis for empirical data exist. This includes application of statistics or hypotheses tests, grounded theory, meta-analysis or systematical reviews, data mining approaches, content analysis, discourse analysis, or qualitative analysis. Most empirical studies today apply either statistical methods or data mining approaches, while few use qualitative analysis of data.

Data Collection: Approaches for gathering data from studies include interviews, questionnaires, data collection / analysis tools (for code analysis; static & dynamic), (manual) collection/analysis of code (static), or audio/video recording. The application of audio or video recording is rare, as this refers to qualitative analysis.

Today, most empirical studies combine controlled experiments with statistical test and some qualitative analysis, usually using data collection tools when examining products, or using manual data collection and analysis otherwise. Some use interviews or questionnaires to address qualitative analysis in experiments or case studies. In particular, we believe that we need to see a wider variety of empirical methods employed, and a stronger emphasis on field studies.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 175 – 177, 2007.

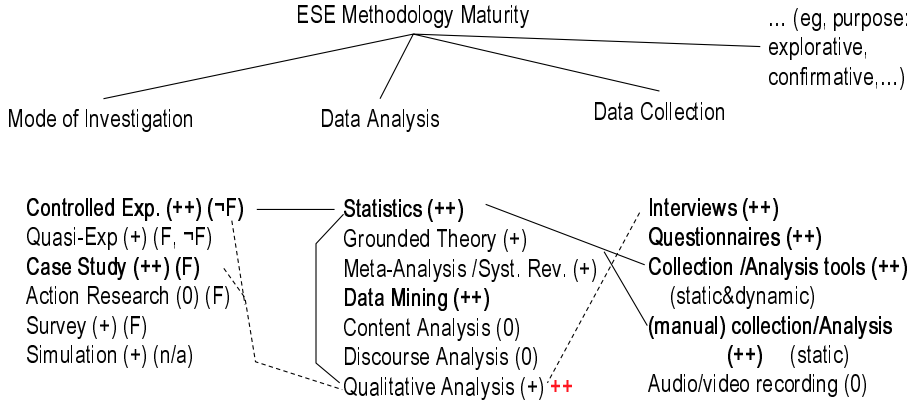


Fig. 1. ESE Methodology maturity refined. In bold: Most common combinations seen in studies today. (++): applied very often (+) applied in some studies; (0) only few applications.

3 Implications for ESE Roadmap

Implications for research can be separated into two areas: Enabling research; that is, research that enhances empirical capabilities, and quantitative increase; that is, more studies of a certain type are needed to increase our knowledge.

3.1 Short-Term (2008)

In the short term, the ESE community needs to improve its opportunity for empirical studies, in particular field studies, for example, through more contact with industry, more lobbying for grants, through targeted (conference) sessions on combining methodologies, through use of practitioner conferences and channels, or through improving empirical education of researchers and practitioners.

In addition, we need to increase the number of studies in some areas; for example, we need to increase the number of studies that combine quantitative and qualitative approaches within case studies and experiments, we need to conduct more experiments in field (quasi-experiments), and in general, more field studies.

That is, in the short term, enabling research is of less importance than application of available empirical capabilities.

3.2 Mid-Term (2009/11)

In the mid term, it is necessary to raise the awareness of the diversity of available techniques, as well as make use of more advanced empirical approaches, such as triangulation, concurrent combination of studies, and approaches to share and build on results.

Again, enabling research as such is not the main concern in the mid-term roadmap, rather awareness and application of the range of available techniques. One exception is the approach to share and build on previous results; in this area, there are currently only few applicable approaches, and further research is necessary.

3.3 Long-Term (2012/14)

In the long term, the challenge in ESE methodology seems to be that we need to be able to define a research program through sequential combination of studies; that is, to be able to define an optimal sequence of studies that can answer a research question. This includes using combination of different types of studies, such as in vivo/vitro, or simulation. A prerequisite for this step in methodology maturity is the adaptation of simulation models into empirical procedures.

Roadmapping
Working Group 3 Results

Frank Houdek

1 Introduction

Commonly agreed roadmaps are an indicator for mature research fields. This also implies that roadmaps are revised and updated on a regular basis. During the Dagstuhl seminar, four parallel working groups addressed this issue. This chapter summarizes the results of one discussion group.

2 Identification of Maturity Indicators

The first step on building/refining a roadmap was to identify indicators of maturity of the research field "empirical software engineering". The next step was then to identify measures to take to achieve progress with respect to some maturity indicators (see Section 3). Figure 1 depicts graphically the identified maturity indicators. They are by no means complete or in-depth well-defined.

The two axis "Topic/Domain Engineering" and "Topic/Domain Management" indicate which areas of software engineering methodology and their interplay has been considered by empirical investigations yet. Current studies usually put emphasis on investigating individual methodologies (e.g. reviews, cost estimation, design patterns). Today, in the engineering area, we see emphasis on studies on analytic techniques. There are fewer studies on constructive technologies, and it is hard to find studies on integrated engineering methodology sets. Similar levels of "challenge" can be identified in the field of management studies. There are quite some studies that investigate software management practices in isolated environments (e.g. cost estimation using space projects). There are less studies that investigate management techniques in multiple environments; integrated studies that consider various management techniques or their interplay with engineering methodologies are hard to find.

The axis "Evidence/Maturity" identifies levels of knowledge that might be achieved over time. Initially, there is a speculative theory. Over time, this might be confirmed by observations; many observations – if well analyzed – may help to identify regularities/rules/common patterns. The ultimate goal in the area of evidence/maturity is a well-defined theory. A well-defined theory as explanatory power, i.e. it can be used to predict behavior in environments that has not yet been investigated before.

The tree axis "Cohesiveness", "Research Group Capabilities" and "Planning of Research Programs/Appropriateness" are field maturity indicators in the narrow sense. They characterize the current capabilities of the research field.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 178 – 180, 2007.

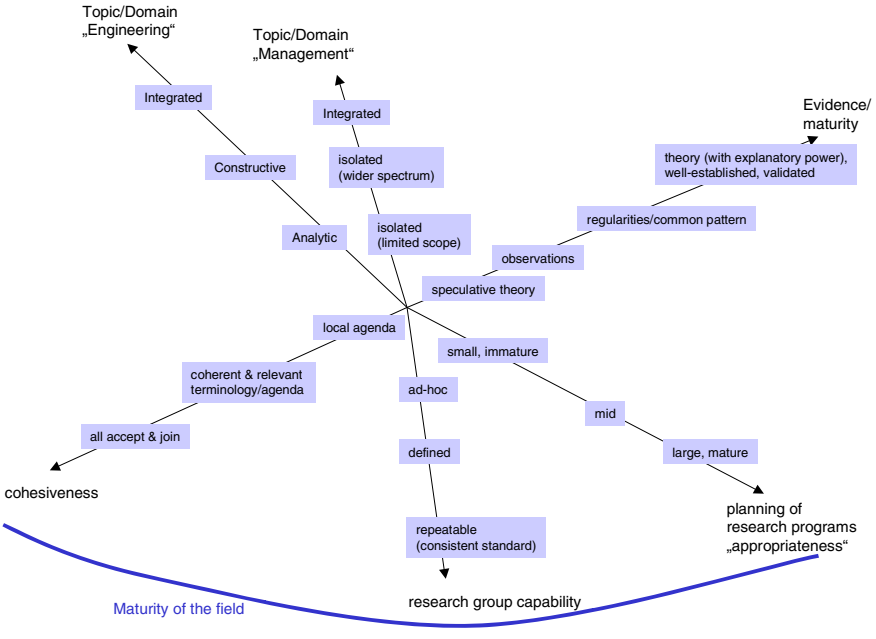


Fig. 1. Dimensions of maturity

“Cohesiveness” characterizes the common understanding – understanding of terminology, relevant research questions, research methodologies, explanatory power of studies and so on. It ranges from local agendas, i.e. each research group identifies locally relevant questions and works on them, but there is no common understanding of relevant questions and mechanisms to work collaboratively on answering these questions. With local agendas, each researcher contributes bricks to a large construction project, but there is no shared plan or vision on the outcome of the construction endeavor.

“Research Group Capabilities” characterizes the way studies are planned, conducted, and analyzed, ranging from ad-hoc methodology to repeatable. Repeatable means that similar research questions are treated with similar (commonly accepted) investigation techniques, the results are independent of the particular research responsible for the study.

“Planning of Research Programs/Appropriateness” characterizes to which extend the used research program is appropriate for the question under consideration.

3 Implications for ESE Roadmap

For three axis (“Cohesiveness”, “Evidence/Maturity”, “Planning of Research Programs/Appropriateness ”) we subsequently identified some short, mid, and long term activities in order to improve the current state with respect the to axis.

Figure 2 shows these activities in a matrix. Again, these findings are by no means complete.

	Evidence maturity	Appropriateness	Cohesiveness
Long term	Access to experiment protocolls --> Review with students	virtual research groups	Landscaping
Mid Term	synthesis papers	evaluation formats	finding coherent terminology
Short term	systematic reviews as standard	intensive collaborations	

Fig. 2. Activities to improve current situation

Identified activities with respect to "Evidence/Maturity" strongly recommend to strengthen the concept of assembling bricks to larger entities instead of adding new bricks to the pile, only.

Conducting an appropriate study is hard. Often it requires to evaluate a certain set of technologies in multiple environments, under varying conditions, in different combinations. Especially investigating integrated topics (see axis "Topic/Domain") requires large studies and thus the intense collaboration of numerous researchers. Virtual research groups or virtual research labs seem to be the appropriate way for this.

An important element in achieving more cohesiveness is to better understand the current situation. There are already literally hundreds of empirical studies, each one contributing in some way to a big picture. But there is no such picture available. Drawing at least larger fragment of such a picture, i.e. landscaping the field, is a most relevant activity. If these landscapes become commonly accepted and every member of the empirical community feels to be responsible to contribute to this map of knowledge, this would significantly help to harmonize research agendas.

Roadmapping
Working Group 4 Results

Laurie Williams, Hakan Erdogmus, and Rick Selby

Our group[1] carefully considered the factors involved with the maturity of the field of Empirical Software Engineering (ESE). A graphical representation of the consensus of the group can be found in the polar chart of Figure 1. Each of the six axes represents a factor. Milestones points in the progress of the factors are delineated on each axis whereby a milestone indicates that the majority of work in ESE is at that level. The distances between the milestones are not drawn to scale. By this, we mean that the progress to move linearly along an axis is not proportional to the amount of work that must take place.

A point on each of the axes is joined via a line. The resulting shape of these lines is our indication of the current state-of-the practice of ESE. In general, the point the furthest away from the center of the chart indicates the ultimate goal in a mature field of ESE. However, due to their nature, the ultimate goal may be an interim point for certain families of study/domains. Each of the factors and the milestones will be discussed below, beginning with Domain/Coverage and progressing clockwise around the polar chart.

Domain/Coverage. Research results are only valid in the context in which the research was conducted. For this reason, we cannot assume *a priori* that the results of a study generalize beyond the specific environment in which it was conducted [1]. Researchers become more confident in a theory when similar findings emerge in different contexts [1]. Initial research results generally involve an isolated study of one artifact. Progression occurs with the examination of multiple artifacts in a system and then to multiple artifacts on a variety of systems in one domain. This research is then replicated in multiple domains. Finally, the family of empirical studies on a topic has been replicated in and/or ported to a comprehensive and representative set of domains.

Evidence Maturity. Researchers begin a line of research with a conjecture (or initial theory) which predicts an outcome. Empirical analyses are conducted to determine if the conjecture can be supported or refuted via observations. A set of analyses are then examined to extract common patterns from the results. The ultimate goal is to obtain a validated theory that is predictive or causal.

Research Methodology Maturity. We consider four general classes of techniques: *in vitro* (research conducted in a laboratory), *in vivo* (research conducted in a live setting), *in virtuo* (subjects interact with a simulated environment) and *in silico* (both subjects and objects are simulated). Initial research results will likely begin with one of these techniques. Ultimately, a respected empirical evaluation will contain results of an integration of all four of these techniques. We also consider a progression from mono-method qualitative or quantitative techniques to multi-method, combinations of qualitative and quantitative techniques.

[1] The group consisted of Hakan Erdogmus, Natalia Juristo, Marek Leszak, Sandro Morasca, Rick Selby, Elaine Weyuker, Laurie Williams, Claes Wohlin, Sira Vegas, Marv Zelkowitz.

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 181 – 183, 2007.

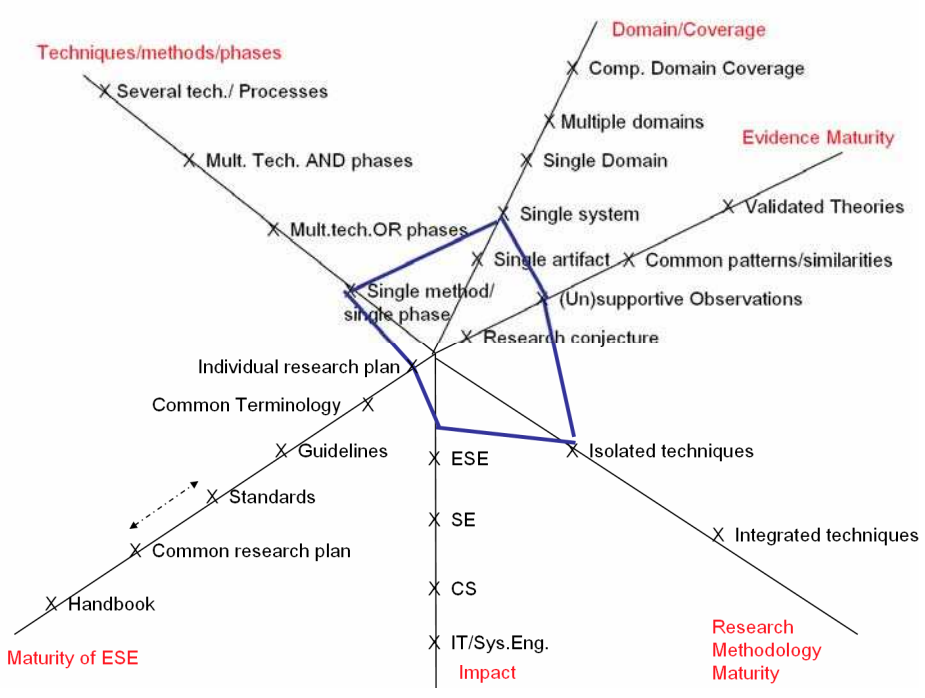


Fig. 1. Empirical Software Engineering Roadmap by Group #4

Impact. The ultimate goal of software engineering research is to impact the practice of software engineering. As the field of ESE matures, its practices will be progressively integrated and accepted by researchers in the field of ESE, then by researchers in software engineering in general, then by researchers in computer scientists, and finally by researchers in the industrial fields of information technology and systems engineering.

Maturity of ESE. Several artifacts must be developed and understandings must evolve for the field of ESE to be mature. As far as artifacts, first an agreed-upon glossary of terminology must be developed and approved by the community. This can be spearheaded by a small working group, whose output can then be reviewed and commented on by the larger community. Using this common terminology, researchers in the community should produce and publish guidelines on ESE practices which can be commented on and evolved by others in the community. A potential venue for developing and publishing these guidelines is a dedicated track of the Empirical Software Engineering journal. Over time, these guidelines can evolve to a set of accepted and published standards. The set of these standards can then be published in a handbook also containing the glossary of terminology. Simultaneously, the field will evolve from the point that every researcher/research group has his/her/their own research agenda to a point where the field has a set of known research agendas and has strategies for jointly tackling relevant topics to produce a validated set of theories in a comprehensive set of domains.

Techniques/methods/phases. In the simplest case, research will focus on a particular technique in one particular development phase (e.g. inspections during the coding phase). Next, this technique will be studied in a variety of phases (e.g. requirements inspections, design inspections, code inspections, test plan inspections). The research could at the next step extend to a comprehensive set of techniques across multiple phases of development. Ultimately, the research will involve several techniques and/or entire processes.

Reference

[1] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge Through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, 1999, pp. 456 - 473.

Empirical Software Engineering Research Roadmap
Discussion and Summary

Richard W. Selby

1 Roadmap Discussions

The Dagstuhl working groups' discussions provide many insightful perspectives and suggestions for defining a roadmap for Empirical Software Engineering (ESE) research. This summary attempts to consolidate these ideas into an overall roadmap. As emphasized in the roadmapping introduction, defining a roadmap is an ongoing process and the resulting roadmap needs to be considered a "living document." New ideas and changing environments will continue to influence the roadmap, and consequently, the roadmap will need to be updated periodically to incorporate these new ideas and environments.

2 Roadmap Categories, Dimensions, and Progress Indicators

Each roadmap dimension defines one important aspect of ESE research. The overall ESE roadmap consists of four categories that organize and group together nine different dimensions. The ESE roadmap categories and dimensions are as follows with dimensions indented below categories:

- Maturity
  - Cohesiveness of field
  - Research methodology
- Coverage
  - Process/technique/phase
  - Problem domain
  - Artifact scale
  - Subject expertise level
- Understanding
  - Evidence
- Impact
  - Science/engineering
  - Industry

Each of the nine dimensions has several "signposts" or indicators that signify progress along the dimension, and each dimension has a final progress indicator that defines the ultimate goal for the dimension. The ordering of these progress indicators suggests a logical path or maturation along a dimension. In some cases, researchers may pursue many steps in parallel so the definition of a strict linear ordering of steps

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 184 – 187, 2007.

can be challenging. The overall ESE roadmap consisting of four categories, nine dimensions, and numerous progress indicators is outlined as follows with progress indicators indented below dimensions and listed in ascending order of progress:

- Category: Maturity
  - Dimension: Cohesiveness of field
    - Individual research plan
    - Common terminology
    - Guidelines
    - Standards
    - Common research plan
    - Handbook
  - Dimension: Research methodology
    - Subjective views
    - Isolated techniques
    - Understand technique tradeoffs
    - Integrated techniques
    - Repeatable methods
    - Objective framework, standards
- Category: Coverage
  - Dimension: Process/technique/phase
    - Single technique/phase
    - Multiple techniques or phases
    - Multiple techniques and phases
    - Comprehensive processes, techniques, and phases
  - Dimension: Problem domain
    - Single artifact
    - Single project
    - Single domain
    - Multiple domains
    - Comprehensive domain coverage
  - Dimension: Artifact scale
    - Units/components
    - Subsystems
    - Small-scale systems
    - Large-scale systems
    - System-of-systems
  - Dimension: Subject expertise level
    - Junior expertise
    - Intermediate expertise
    - Advanced expertise
- Category: Understanding
  - Dimension: Evidence

    - Research conjectures
    - Supportive observations
    - Common patterns/similarities
    - Replicated results
    - Validated theories
- Category: Impact
  - Dimension: Science/engineering
    - Subset of Empirical Software Engineering
    - Empirical Software Engineering
    - Software Engineering
    - Computer Science
    - Systems Engineering and/or Information Technology
  - Dimension: Industry
    - Isolated examples
    - Organizational adoption
    - Multi-organizational adoption
    - Industry-wide shift

3 Roadmap Visualization

Figure 1 displays a Kiviat-style or "spiderweb" graph of the ESE roadmap. Researchers can assess progress along each dimension by evaluating research activities and results according to each dimension's progress indicators. An overall assessment of the ESE field emerges when researchers assess all the dimensions simultaneously. One such combined assessment calibrates the ESE field as follows:

Dimension	Current Progress Indicator
Cohesiveness of field	Individual research plan
Research methodology	Isolated techniques
Process/technique/phase	Single technique/phase
Problem domain	Single project
Artifact scale	Units/components
Subject expertise level	Junior expertise
Evidence	Supportive observations
Science/engineering	Subset of Empirical Software Engineering
Industry	Isolated examples

Future workshops will need to update the proposed ESE roadmap and reassess progress. ESE continues to be a very fruitful area for rich software engineering research, and we look forward to continued progress in the field.

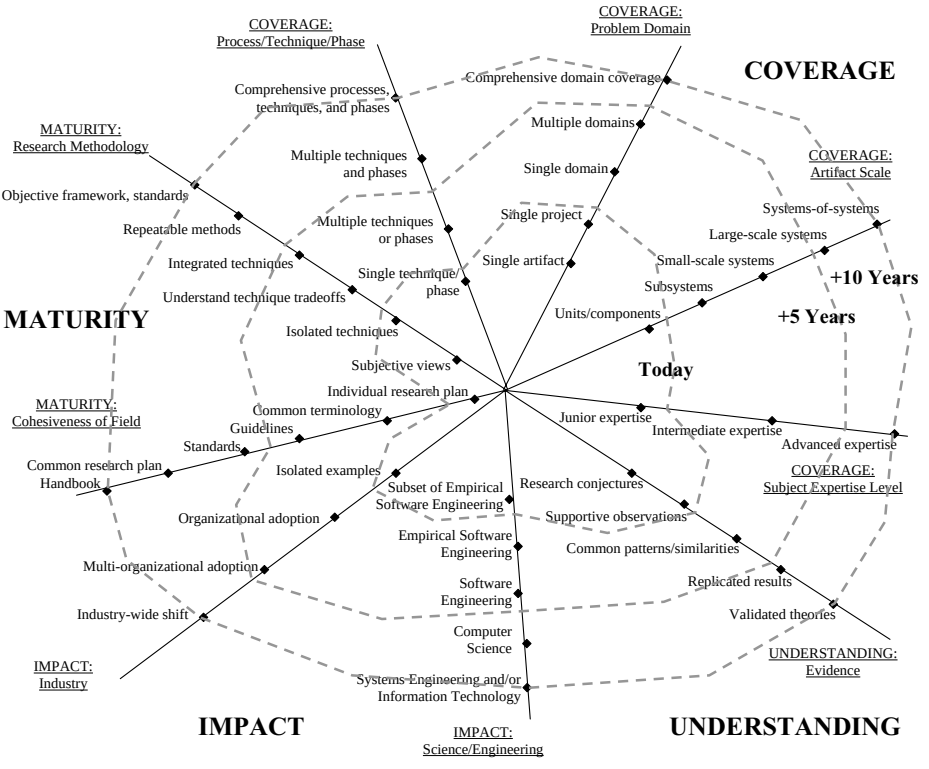


Fig. 1. Empirical Software Engineering Research Roadmap

List of Participants

Organizing Committee

Victor Basili
Univ. of Maryland at College Park
Dept. of Computer Science
Room 4111
A.V. Williams Bldg.
MD 20742 College Park, (USA)
Fax: +1-301-405-3691
E-Mail: basili@cs.umd.edu

Dieter Rombach
Fraunhofer Institut- Experimentelles Software Engineering (Fh IESE)
Fraunhofer Platz 1
D-67663 Kaiserslautern, (D)
Tel: +49-631-6800-1000
Fax: +49-631-6800-1099
E-Mail: rombach@iese.fraunhofer.de

Kurt Schneider
Universität Hannover
Institut für Praktische Informatik
FG Software Engieering
Welfengarten 1
D-30167 Hannover, (D)
Tel: +49-511-762-19666
Fax: +49-511-762-19679
E-Mail: kurt.schneider@inf.uni-hannover.de

Academic Invitees

Lionel C. Briand
Carleton University
Dept. of Systems & Computer Engineering
Minto CASE Buildings, Room 7082
1125 Colonel By Drive
ON-K1S 5B6 Ottawa, (CDN)
Tel: +1-613-520-2600
Fax: +1-613-520-5727
E-Mail: briand@sce.carleton.ca

Giovanni Cantone
Università di Roma "Tor Vergata"
Dept. of Informatics, Systems and Productioning
DISP - Facoltà di Ingegneria
Via del Politecnico 1
I-00133 Roma, (I)
Fax: +39 06 7259 7460
E-Mail: cantone@uniroma2.it

Jeffrey Carver
Mississippi State University
Dept. of Computer Science and Engineering
P.O. Box 9637
MS 39762-9627 Mississippi State, (USA)
Tel: +1-662-325-0004
Fax: +1-662-325-8997
E-Mail: carver@cse.msstate.edu

Marcus Ciolkowski
TU Kaiserslautern
FB Informatik
Postfach 3049
D-67653 Kaiserslautern, (D)
Tel: +49 (631) 6800-2233
E-Mail: ciolkows@informatik.uni-kl.de

V. Basili et al. (Eds.): Empirical Software Engineering Issues, LNCS 4336, pp. 188 – 192, 2007.

Tore Dybå
SINTEF ICT
S.P. Andersens vei 15B
N-7465 Trondheim, (N)
Tel: +47-73-59-29-47
Fax: +47-73-59-29-77
E-Mail: tore.dyba@sintef.no

Hakan Erdogmus
National Research Council
Institute for Information Technology
M-50 Montreal Road
ON-K1A OR6 Ottawa, (CDN)
E-Mail:
Hakan.Erdogmus@nrc-cnrc.gc.ca

Robert Glass
Griffith University
School of Computing & Information Technology
170 Kessels Road
Nathan
QLD 4111 Brisbane, (AU)
E-Mail: rlglass@acm.org

Tracy Hall
University of Hertfordshire
Department of Computer Science
College Lane
AL10 9AB Hatfield, (GB)
Fax: +44 1707 28 4303
E-Mail: t.hall@herts.ac.uk

Martin Höst
Lund University
Dept. of Communication Systems
Box 118
SE-22100 Lund, (S)
Tel: +46-46-222 9016
Fax: +46-46-145 823
E-Mail: martin.host@telecom.lth.se

Andreas Jedlitschka
Fraunhofer Institut
IESE - Experimentelles Software Engineering
Fraunhofer-Platz 1
D-67663 Kaiserslautern, (D)
Tel: +49-631-6800-2260
Fax: + 49-631-6800-1599
E-Mail: jedlitschka@iese.fraunhofer.de

Ross Jeffery
National ICT Australia
ESE
Locked Bag 9013
NSW 1435 Alexandria, (AU)
Tel: +61 2 8374 5512
Fax: +61 2 8374 5520
E-Mail: rossj@cse.unsw.edu.au

Natalia Juristo
Universidad Politecnica de Madrid
Facultad de Informática
Campus de Montegancedo
E-28660 Boadilla del Monte, Madrid, (E)
Tel: +34 91336 6922
Fax: +34 91336 6917
E-Mail: natalia@fi.upm.es

Barbara Kitchenham
Keele University
Dept. of Computer Science
ST5 5BG Staffordshire, (GB)
Tel: +44-1782 583413
Fax: +44-1782 713082
E-Mail:
barbara.kitchenham@cs.keele.ac.uk

Michael S. Mahoney
Princeton University
Department of History
303 Dickinson Hall
NJ 08544 Princeton, (USA)
Tel: +1-609-258-4157
Fax: +1-609-258-5326
E-Mail: mike@princeton.edu

James Miller
University of Alberta
Electrical & Computer Engineering
Research Dept.
2nd Floor - ECERF - 9107 - 116 Street
T6G 2V4 Edmonton AB, (CDN)
Tel: +1-780-492-1181
Fax: +1-780-492-6153
E-Mail: jm@ee.ualberta.ca

Sandro Morasca
Università dell'Insubria della Cultura, Politiche
Dipartimento di Scienze e dell'Informazione
Via Valleggio 11
I-22100 Como, (I)
Tel: +39-031-238-6228
Fax: +39-031-238-6119
E-Mail: sandro.morasca@uninsubria.it

Matthias Müller
Universität Karlsruhe
Fakultät für Informatik
50.34 Informatik-Hauptgebäude 372
Postfach 6980
D-76128 Karlsruhe, (D)
Fax: +49-721/608-7343
E-Mail: muellerm@ira.uka.de

Jürgen Münch
Fraunhofer Institut
Experimentelles Software Engineering (Fh IESE)
Fraunhofer-Platz 1
D-67663 Kaiserslautern, (D)
Fax: + 49 631 6800 1301
E-Mail:
juergen.muench@iese.fraunhofer.de

Markku Oivo
University of Oulu
Department of Information Processing Science (TOL)
P.O. Box 3000
FIN-90014 Oulu, (FIN)
Tel: +358-40-822-7702
Fax: +358-8-5531-890
E-Mail: Markku.Oivo@oulu.fi

Dietmar Pfahl
University of Calgary
Dept. of Computer Science & Electrical Engineering
Schulich School of Engineering
2500 University Drive N.W.
T2N 1N4 Calgary, (CDN)
Fax: +1-403-282-6855
E-Mail: dpfahl@ucalgary.ca

Lutz Prechelt
Freie Universität Berlin
FB Mathematik und Informatik
Takustr. 9
D-14195 Berlin, (D)
Tel: +49-30-838-75115
Fax: +49-30-838-75218
E-Mail: prechelt@inf.fu-berlin.de

Austen W. Rainer
University of Hertfordshire
Department of Computer Science
College Lane
AL10 9AB Hatfield, (GB)
Tel: +44-1707-284-763
Fax: +44-1707-284-303
E-Mail: a.w.rainer@herts.ac.uk

Carolyn Seaman
Univ. of Maryland at Baltimore Country (Seaman)
CSEE Dept.
Room 444
1000 Hilltop Circle
MD 21250 Baltimore, (USA)
Fax: +1 410 455 1073
E-Mail: cseaman@umbc.edu

Helen C. Sharp
The Open University
Mathematics & Computing Department
MK7 6AA Milton Keynes, (GB)
Tel: +44-1908-653638
Fax: +44-1908-652140
E-Mail: h.c.sharp@open.ac.uk

Dag Sjøberg
Simula Reseach Laboratory
Software Engineering
Martin Linges Vei 17, Fornebu
Post Box 134
N-1325 Lysaker (Fornebu), (N)
Fax: +47 67 82 82 01
E-Mail: dagsj@simula.no

Walter F. Tichy
Universität Karlsruhe
Inst. für Programmstrukturen und
Datenorganisation (IPD)
Am Fasanengarten 5
Postfach 6980
D-76128 Karlsruhe, (D)
Tel: +49-721-608-3934
Fax: +49-721-608-7343
E-Mail: tichy@ira.uka.de

Guilherme Horta Travassos
Federal University of Rio de Janeiro
Graduate School of Engineering
(COPPE)
PO Box 68511
21945-972 Rio de Janeiro, (BR)
Tel: +55-21-2562-8712
Fax: +55-21-2562-8676
E-Mail: ght@cos.ufrj.br

Sira Vegas
Universidad Politecnica de Madrid
Facultad de Informática
Campus de Montegancedo
E-28660 Madrid, (E)
Tel: +34 91336 6929
Fax: +34 91336 6917
E-Mail: svegas@fi.upm.es

Laurie Williams
North Carolina State University
Computer Science Department
EB2, Room 3272
890 Oval Drive
NC 27695 Raleigh, (USA)
Fax: +1-919-513-1895
E-Mail: williams@csc.ncsu.edu

Claes Wohlin
Blekinge Institute of Technology
Dept. of Systems and Software
Engineering
School of Engineering
SE-37225 Ronneby, (S)
Fax: +46-457-271-25
E-Mail: claes.wohlin@bth.se

Marvin Zelkowitz
Univ. of Maryland at College Park
Dept. of Computer Science
MD 20742 College Park, (USA)
Fax: +1-301-405-3691
E-Mail: mvz@cs.umd.edu

Industrial Invitees

Frank Houdek
DaimlerChrysler AG
Forschung & Technik /
Prozeßgestaltung
Wilhelm-Runge-Str.11
Postfach 2360
D-89081 Ulm, (D)
Tel: +49-731-505-2855
Fax: +49-731-505-4218
E-Mail:
frank.houdek@daimlerchrysler.com

Marek Leszak
Lucent Technologies
Bell Labs MNS Software Development
Thurn-und-Taxis-Str. 10
D-90411 Nürnberg, (D)
Tel: +49-911-526-3382
E-Mail: mleszak@lucent.com

Audris Mockus
Avaya Labs.
SW Technology Research Dept.
233 Mt. Airy Road
NJ 07920 Basking Ridge, (USA)
Tel: +1-908-696-5608
Fax: +1-908-696-5401
E-Mail:
audris@research.avayalabs.com

Nachiappan Nagappan
Microsoft Research
Software Reliability Research (SRR)
One Microsoft Way
WA 98052-6399 Redmond, (USA)
Fax: +1-425-936-7329
E-Mail: nachin@microsoft.com

Tom Ostrand
AT&T Labs Research
E 237
180 Park Avenue
NJ 07932 Florham Park, (USA)
Tel: +1-973-360-8134
E-Mail: ostrand@research.att.com

Rick Selby
Northrop Grumman Space Technology
R4/2011B
One Space Park
CA 90278 Redondo Beach, (USA)
Tel: +1-310-813-5570
Fax: +1-310-814-4941
E-Mail: rick.selby@ngc.com

Lawrence G. Votta
Sun Microsystems
CARE; MS MPK17-119; Room 1374
17 Network Circle
CA 94025 Menlo Park, (USA)
Tel: +1-425-829-2786
Fax: +1-650-786-2328
E-Mail: larry.votta@sun.com

Elaine Weyuker
AT&T Labs Research
C213
180 Park Avenue
NJ 07932 Florham Park, (USA)
Tel: +1-973-360-8645
Fax: +1-973-360-8077
E-Mail: weyuker@research.att.com

Author Index