Pekka Abrahamsson
Michele Marchesi
Frank Maurer (Eds.)

LNBIP 31

Agile Processes in Software Engineering and Extreme Programming

10th International Conference, XP 2009
Pula, Sardinia, Italy, May 2009
Proceedings

Springer

Pekka Abrahamsson Michele Marchesi
Frank Maurer (Eds.)

Agile Processes in Software Engineering and Extreme Programming

10th International Conference, XP 2009
Pula, Sardinia, Italy, May 25-29, 2009
Proceedings

Springer

Volume Editors

Pekka Abrahamsson
University of Helsinki
Department of Computer Science
Helsinki, Finland
E-mail: Pekka.Abrahamsson@cs.helsinki.fi

Michele Marchesi
University of Cagliari
DIEE Department of Electrical and Electronic Engineering
Cagliari, Sardinia, Italy
E-mail: michele@diee.unica.it

Frank Maurer
University of Calgary
Agile Software Engineering/e-Business Engineering (ase/ebe) group
Calgary, Canada
E-mail: frank.maurer@ucalgary.ca

Library of Congress Control Number: Applied for

ACM Computing Classification (1998): D.2, K.6

ISSN 1865-1348
ISBN-10 3-642-01852-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01852-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12681709 06/3180 5 4 3 2 1 0

Preface

The field of software engineering is characterized by speed and turbulence in many regards. While new ideas are proposed almost on a yearly basis, very few of them live for a decade or a longer. Lightweight software development methods were a new idea in the latter part of the 1990s. Now, ten years later, they are better known as agile software development methods, and an active community driven by practitioners has formed around the new way of thinking. Agile software development is currently being embraced by the research community as well. As a sign of increased research activity, most research-oriented conferences have an agile software development track included in the conference program.

The XP conference series established in 2000 was the first conference dedicated to agile processes in software engineering. The idea of the conference is to offer a unique setting for advancing the state of the art in research and practice of agile processes. This year's conference was the tenth consecutive edition of this international event. Due to the diverse nature of different activities during the conference, XP is claimed to be more of an experience rather then a regular conference. It offers several different ways to interact and strives to create a truly collaborative environment where new ideas and exciting findings can be presented and shared. This is clearly visible from this year's program as well. Looking at the conference program you can see a large variety of different types of contents ranging from regular full research papers to tutorials, workshops and other activities, which are designed to stimulate participant collaboration.

The XP conference continues to increase its academic standing year by year. The XP Committee will seek to build upon this trend in the coming years. The XP paper submissions went through a rigorous peer-review process. Each paper was reviewed by at least three Program Committee members. Of 40 papers submitted, only 12 were accepted as full papers (30%). The papers represent a set of high-quality research studies addressing a wide variety of different topics ranging from deep technical issues to a wide range of human issues in agile software development including novel ideas on agile software research. Besides the regular conference activities, the program also featured a conference-within-a-conference open space event that allowed participants to hold discussions on late-breaking topics on the fly. This year's XP also presented a number of high-profile keynotes from Mary Poppendieck, Bjarte Bogsnes and Ivar Jacobson. XP 2009 also included the largest workshop offering ever seen in the conference's history. The participants had the option to participate in more than 20 workshops and 16 tutorials.

We would like to extend our gratitude to all those who contributed to the organization of XP 2009. The authors, the sponsors, the Chairs, the reviewers, and all the volunteers: without their help, this event would not have been possible

March 2009

Pekka Abrahamsson
Frank Maurer
Michele Marchesi

Organization

Conference Chairs

General Chair

Michele Marchesi	University of Cagliari, Sardinia, Italy

Program Chairs

Pekka Abrahamsson	University of Helsinki
Frank Maurer	University of Calgary, Canada

Local Organizing Chairs

Giulio Concas	University of Cagliari, Italy
Sandro Pinna	Sardegna IT, Italy

Tutorial Chairs

Rachel Davies	Agile Experience Limited, UK
Philippe Kruchten	University of British Columbia, Canada

Workshop Chairs

Charlie Poole	Poole Consulting, USA
Jutta Eckstein	IT Communication, Germany
Torgeir Dingsøyr	SINTEF ICT, Norway

Special Events Chair

Fraser	Cisco Research ,USA

Poster Chairs

Alberto Sillitti	Free University of Bozen Bolzano, Italy
Pasi Kuvaja	University of Oulu, Finland

PhD Symposium Chairs

Karlheinz Kautz	Copenhagen Business School, Denmark
Kieran Conboy	NUI Galway, Ireland

Panel and Activities Chair

Steven Freeman	M3P, UK

Tools and Demonstration Chairs

Daniel Kalström	Golden Gekko, Sweden
Werner Wild	Evolution, Austria

Open Space Chairs

Willem van den Ende	willemvandenende.com, The Netherlands
Lasse Koskela	Reaktor Innovations, Finland

Sponsorship Chairs

Ko Dooms	Philips, The Netherlands
Jari Still F	Secure, Finland

Publicity Chairs

Europe: Minna Pikkarainen	VTT Technical Research Centre of Finland
Oceania: Nils Brede Moe	SINTEF ICT, Norway
America: J.B. Rainsberger	Canada
Middle East: Ahmed Sidky	X2A Consulting, USA
India: Naresh Jain	India

International Student Volunteer Chair

Noura Abbas	University of Southampton, UK

Official Photographers

Tom Poppendieck	Poppendieck.LLC, USA
Hubert Baumeister	Technical University of Denmark, Denmark

Web Chair

Enrico Marongiu	Sardegna IT, Italy, enrico@xp2009.org

Program Committee

James D. Arthur
M. Ali Babar
Richard Baskerville
Robert Biddle
Stefan Biffl
Gerardo Canfora
Joseph Chao
Mike Chiasson
Yael Dubinsky
Tore Dyba
Hakan Erdogmus
Tor Erland Faegri
Jennifer Ferreira
Juan Garbajosa
Yasser Ghanam
Paul Gruenbacher
Geir Kjetil Hanssen
Orit Hazzan
Ron Jeffries
Kari Känsala

Frank Keenan
Mikko Korkala
Maarit Laanti
Stig Larsson
Kalle Lyytinen
Mary Lynn Manns
Angela Martin
Fergal McCaffery
Grigori Melnik
Sridhar Nerur
Dave Nicolette
Peter Axel Nielsen
Markku Oivo
Shelly Park
Joseph Pelrine
Gary Pollice
Daniel Port
Agerfalk Par
Knut Rolland
Barbara Russo
Outi Salo
Helen Sharp
Park Shelly
Giancarlo Succi
Tom Tourwe
Corrado Aaron Visaggio
Xiaofeng Wang
Barbara Weber
Don Wells
Laurie Williams
Hongyu Zhang

Table of Contents

What They Dont Teach You about Software at School: Be Smart!

Ivar Jacobson

Ivar Jacobson International
ivar@ivarjacobson.com
http://www.ivarjacobson.com

Abstract. One of the most popular buzzwords in software development is agile. Today everyone wants to be agile. That is good! However, being agile is not enough.You also need to be smart. What does that mean? Smart is about being agile, but it is also about doing the right things, the right way. You can become smarter through training. However, without experience your alternatives are too many and only a few of them are smart. Experience is of course something you can get sooner or later, but it takes time if you must learn by your own mistakes. This is where the utilization of "smart-cases" becomes essential. In this talk, we will describe a large number of smart-cases when developing software. It is about working with people, teams, projects, requirement, architecture, modeling, documentation, testing, process, and more.

Keywords: Agile software development.

1 What Does It Mean to Be Smart?

The essence of being agile is being smart. I have for several years expressed that the most important character you need to have to be a great software developer is to be smart. In several of my columns I have summarized what you need to do to be successful by saying: You need to be smart! What does that mean? Most people know intuitively what "being smart" means in everyday language, but what does it mean for software.

"Things should be done as simple as possible but no simpler" to quote Albert Einstein and that expression says in a nutshell what it means to be smart.

Smart is not the same thing as being intelligent. You can be intelligent without being smart. I know lots of people who are intelligent but not smart. You can be very smart without being very intelligent.

Smart is not the same as having common sense. You can have common sense without being smart, but if you are smart you must however have common sense.

Being Smart is an *evolution* of being agile, but it is also about doing the right things, the right way. With a popular notion you may say that smart = agile++.

2 What They Don't Teach You at Institutes of Technology

Interestingly, many professors in software have never built any useful software. This is natural if they have grown up in the academic world where software engineering is not as fine as computer science. Thus, many professors cant possible teach software engineering with any passion.

As a consequence new ideas about software engineering have come from practitioners around the world. The best ideas come from practitioners with an appreciation of science, people that agree with Kurt Lewin: “There is nothing as practical as a good theory”. Examples of good theories were Newtons laws, and Einsteins relativity theory. A lot of research represents useless theories.

Personally, I adopted this motto thirty years ago and it has been leading my work all these years: components, use cases, Objectory that became RUP, SDL and UML, and now practices. Many other people such as Kent Beck and Ken Schwaber have made similar contributions. This is the true essence of software engineering and it stands on a sound theoretical basis, meaning it can be explained so others can understand it and practice it. But, few universities teach it and even fewer do it well.

However, these exceptional institutes that teach you software engineering practices rarely would teach you how to be *smart* when working with these practices. And this is fine. I am happy if they teach good practices, but to provide real value, each practice needs to be applied in a *smart* way, otherwise you will do too much or too little. This is where the utilization of what we call “smart cases” becomes essential. We have a large number of smart cases that relate to working with people, teams, projects, architecture, modeling, documentation, testing, process, outsourcing, and many more.

3 Smart Cases

Each smart case has a subject, a context, a typical case dealt with in an unsmart way and in a smart way, and a key motto for how to be smart.

The mottos are critical, so let us start with some examples:

- People: Software is developed by people, not by process and tools.
- Teams: A software team is like a sport team with all needed competencies to win.
- Project: Think big, build in many steps.
- Architecture: Start to build a skinny system, add muscles in later steps.
- Testing: Whatever you do you are not done until you have verified that you did what you wanted to do.
- Documentation: Focus on the essentials - the placeholders for conversations people figure out the rest themselves.
- Process: Dont throw out what you have, instead start improving from your existing way of working, find your pain points and change one practice at the time.

And on it goes.

The unsmart/smart relationship helps us understand the balance. Lets take two subjects, Project and Architecture:

Project
It is not smart to work in a waterfall manner, first specify all requirements, then do the design and the code, and finally test it all. You will create a lot of paper-ware but you wont know if this is what you want to build, you dont know if it is testable, and you will discover serious problems with performance, architecture, usability too late. It is smart to first build a small skinny system and then build a little bit more, and a little bit more, before you release the system to customers. Each time you build something, you must be able to run, validate and test it.

Architecture
It is not smart to model everything in UML (happened often for enterprise architecture projects). It is not smart to model nothing and go straight to code. It is however smart to find exactly that something that is of importance to model and code.

What is that something? It is about the most essential use cases and in particular about the most essential scenarios through these use cases. It is about the components and in particular the parts of those components that realize these essential scenarios. Thus, it is about the essentials. Now you may ask what makes a scenario essential. An essential scenario is the response to the question: "what are the most important scenarios the system is supposed to do". Which scenarios exercise the critical parts of the architecture? Which scenarios need to work in order for us to say that the highest technical risks have been eliminated?

In general it is not smart to be extreme in what you do such as: model everything or model nothing, follow a strict waterfall process or an unstructured iterative approach, throw out what you have and start all over. It is smart to be balanced to do what is needed right now but with an eye to the future.
How do you become smart?

I really think being smart is at the core of being successful in software. The answer depends on who you are. Lets assume you are a leader of some sort, for instance a project manager.

To become smart you need first and foremost to have knowledge and secondly to have experience applying that knowledge. You need knowledge to understand what software development is all about. You need to know how to get good software, quickly and at low cost. It is as simple as that <grin>.

Good software is useful (obvious but not trivial), reliable (works always) and extensible (can grow as long as you want to use it). You need to know how you create good software which means you need to learn some technical practices such as architecture, use cases and iterations/sprints.

Nothing makes you as quick (and agile) as having motivated people. You get this by adopting proven agile practices such as how you work as a team, how you organize your work, etc. We all need to be agile.

The most powerful way to get software at low cost is by developing as little software as possible. Instead you acquire the many pieces of software you need from other sources. This is called software reuse. You can reuse your companys old software, buy software from a vendor or download open source software. You need to know how to put it all together with some glue and how to only develop what is really needed. Thus you need some other more advanced practices such as executable enterprise architecture, service-oriented architecture or product-line engineering.

Thus you need knowledge on practices.

You also need experience in using these practices. This is not trivial. Experience comes from having seen a lot of what works and what does not, either directly or through listening to others.This is a subtle (I hope) plug for what we do - we work with many different projects and we gain insight into what works and doesn't. But it's also a plug for communities, for being open to new ideas and experiences, and to strive to be continuously learning. There is a saying that good judgment comes from experience, most of which comes from bad judgment. Except that you don't need to repeat the mistakes of others to learn.

All of this to say, we should learn software engineering practices at school. We should shorten the time to get experience of practices by collecting reusable smart cases. We have started this. We hope others will contribute. I think that would be smart by the industry.

Experience makes you smart through "focus" - knowing what to focus on and what can be ignored. It's not the amount of effort applied, but knowing where and when to apply that effort.

4 Final Words

You may now ask, given that you have knowledge of important practices and you have experience: "Will I now become smart?"

Of course eventually, it comes back to you. We cannot all become equally smart but we can all become smarter.

Keynote: Beyond Budgeting in a Lean and Agile World

Bjarte Bogsnes

Vice President, Performance Management Development, StatoilHydro, Norway
BJBO@StatoilHydro.com

Abstract. We suggest that we need a fundamentally different philosophy and a concept for how organisations are led and managed. We believe that it is time to challenge the traditional management myths. The main goal of Beyond Budgeting is not get rid of budgets. The budget must go, but it is more the budgeting mindset we need to get rid of, because it represents the old thinking that needs to be changed. Based on the works of The Beyond Budgeting Round Table, we review a set of 12 principles for a new style of contemporary leadership and management.

Keywords: Beyond budgeting, lean development, agile development, leadership.

1 Introduction

The world around us has changed. Everything seems to happen faster and more unexpected than before. There is dramatically more uncertainty and change out there, and it started long before the financial crisis. Understanding what might lie around the next corner was much easier when I started my finance career in the corporate budgeting(!) department of the Norwegian oil company Statoil, back in 1983.

There will be more change in the future. Companies are shifting from mass production orientation to knowledge intensive organisations where the human capital is the most valued asset. Companies need to realise that their people are actually highly competent and mature, and expect to be empowered, appreciated, and well - treated as adults. People both want and can take responsibility. I hardly know anyone who wakes up in the morning and goes to work with a firm determination of trying to do a bad job. I believe most people want to do their best. Unfortunately, much of traditional management and leadership practices in companies often work as a barrier to high performance rather than a support.

In this new environment, we need to respond faster and think differently about what best motivates and drives great performance in organisations. The Beyond Budgeting movement believes that the answer to these important questions lies in a fundamentally different philosophy and a concept for how organisations are led and managed. We believe that it is time to challenge the traditional management myths. We must let go of the centralised “command-and-control” thinking, built on a negative “theory X” view on people. We also argue that it is time to challenge the traditional beliefs about control as well. Rather than a control, current management practices often create an illusion of control. Indeed, our controls may seem to provide comfort in uncertain times, but too often they do just the opposite of what they are intended for. We go so far as to claim that they systematically destroy motivation and performance on a scale few have understood so far.

2 Beyond Budgeting: The New Mindset

The main goal of Beyond Budgeting is not get rid of budgets. The budget has to go, but it is more the *budgeting mindset* we need to get rid of, because it represents so much of the old thinking, which we suggest it is time to leave behind.

In addition, we find that the budgets can also destroy the *quality* of what they are meant to provide; good targets, reliable forecasts and an efficient resource allocation. We argue that these three purposes cannot be combined in one process resulting in one number. They also represent *waste* by consuming far too much time and energy. Waste is produced first in the making of annual, detailed budgets and later in the reporting and deviation analysis phase.

An increasing number of companies see the same problems and are exploring alternatives to budgeting. The pioneer is the Swedish bank Handelsbanken, which completely abandoned budgeting in 1970. The bank has constantly been among the best performing banks in Scandinavia since then, and is also the most cost effective universal bank in Europe. Other examples include companies like Toyota, Google, W.L. Gore, Aldi, American Express, WholeFoods and Southwest Airlines, just to name a few. It is not a coincidence that many of these also are Lean and Agile pioneers, because the underlying philosophy is very much the same.

3 Beyond Budgeting: Leadership Principles

The Beyond Budgeting Round Table[1] (BBRT) was established in 1998. This is a global network of companies seeing similar problems with traditional management practices. BBRT has developed the following twelve principles, which together provide a robust and coherent alternative to traditional ways of leading and managing an organisation.

Leadership Principles

- **Customers.** Focus everyone on improving customer outcomes, **not** *on hierarchical relationships.*
- **Organization.** Organize as a network of lean, accountable teams, **not** *around centralized functions.*
- **Responsibility.** Enable everyone to act and think like a leader, **not** *merely follow the plan.*
- **Autonomy.** Give teams the freedom and capability to act; **do not** *micromanage them.*

[1] The present author currently acts as the chairman of the BBRT.

- **Values.** Govern through a few clear values, goals, and boundaries, **not** *detailed rules and budgets.*
- **Transparency.** Promote open information for self-management; **do not** *restrict it hierarchically.*

Process Principles

- **Goals.** Set relative goals for continuous improvement; **do not** *negotiate fixed performance contracts.*
- **Rewards.** Reward shared success based on relative performance, **not** *on meeting fixed targets.*
- **Planning.** Make planning a continuous and inclusive process, **not** *a top-down annual event.*
- **Controls.** Base controls on relative indicators and trends, **not** *on variances against plan.*
- **Resources.** Make resources available as needed, **not** *through annual budget allocations.*
- **Coordination.** Coordinate interactions dynamically, **not** *through annual planning cycles.*

The importance of each principle, and the order to address them in, may vary across companies depending on their business, history and culture. What is critical, however, is that *both* the leadership and the process aspect are addressed in order to achieve a sustainable change. All elements in the management model need to support the same philosophy. It does not help to have great team values if the reward principles only praise individual performance.

4 Way to the Future

Several communities and professions are waking up and rebelling against the old management myths, for instance in the finance sector and within information technology. We suggest that it is time to join forces to secure that lean and agile thinking become the foundation for running the successful 21st century organisation.

My personal journey of heading up two Beyond Budgeting projects, first in Borealis and later in Statoil is described in my new book (See [1] for details).

References

1. Bogsnes, B.: Implementing Beyond Budgeting – Unlocking the Performance Potential. Wiley, Chichester (2008) ISBN : 978-0-470-40516-1
2. Hope, J., Fraser, R.: Beyond Budgeting. Harvard Business School Press, Boston (2003) ISBN 1-57851-866-0

Developing a Test Automation Framework for Agile Development and Testing

Eunha Kim, Jongchae Na, and Seokmoon Ryoo

NHN Corporation, Venture Town Bldg., 25-1 Jeongja-dong,
Seongnam City, Kyeonggi-do, 463844, South Korea
{eunha.kim,monster,seokmoon.ryoo}@nhncorp.com

Abstract. As software developers today, we all face problems of repetitive and error-prone processes, a lack of a clear way of communication between stakeholders, and risks of late defect discovery or release delays. In order to help solve such problems, we implemented an effective framework for automated testing, which combines the automation features of STAF/STAX and the ease-of-use based on tabular input and output of FitNesse. This framework can support Continuous Integration as an automated testing framework to improve software development processes. The greatest advantage of the framework is the agility that allows for rapid delivery of high-quality software. In this paper, we describe the practices and benefits of using the proposed framework.

Keywords: Test Automation Framework, Automated Testing, Agile Testing, Continuous Integration, STAF, STAX, Fit, FitNesse.

1 Introduction

A growing trend in software development is to use testing automation tools which dramatically improve productivity. Despite increasing automation solutions, a significant amount of man-hours are still spent performing manual tests for all types of testing such as new functionality, new platforms, or due to insufficient time and/or skills to develop test scripts. In order to help solve such problems, we implemented an improved framework for automated testing through the integration of FitNesse and the Software Testing Automation Framework (STAF).

Fit [1] is an automated testing framework, developed by Ward Cunningham, to enhance collaboration in software development. FitNesse is a Wiki built on top of the Fit framework which is used for automating acceptance test cases. Both Fit and FitNesse enhance the collaboration between customers, testers and programmers by providing a single platform for communication. Yet it does not allow remote execution. STAF [2] is an open source, multi-platform, multi-language framework designed around the idea of reusable components called services. STAF does allow distributed execution, meaning that from a STAF control machine, tests can be executed that will be sent to either the client or server machines, executed locally on the machine and then returned back to the control machine for reporting. It is designed to be used as pluggable STAX (STAf eXecution engine) for automated test case distribution, execution, monitoring and results analysis. From what I have seen of STAF, writing in xml/python can be a bit messy, and although most of the python code can be extracted into libraries, we would still have to deal with the xml side of things. We wondered if we would be able to obtain a clear overview of the test design and/or plan.

Combining FitNesse and STAF is a powerful mix and enables us to communicate with various stakeholders easily, because the workflow of tests and test environments are graphically expressed into tables of input data and expected output data. Picking up new builds automatically, installing them on remote machines, and executing tests remotely can be accomplished by sending commands via STAF to the applications and reporting the results back in FitNesse. This framework helps automate the distribution, execution and results analysis of test cases [2]. It also aids communication among the various stakeholders, using tables for representing tests and for reporting the results of automatically checking those tests [1]. We first described details of the proposed framework in a paper presented at ITNG2009 [3].

2 NTAF

The framework described in this paper is referred to as the NHN Test Automation Framework (NTAF). Fig. 1 depicts the architecture of NTAF. NTAF provides multiple benefits by automating repetitive and error-prone processes. In this section we explain the powerful advantages of NTAF.

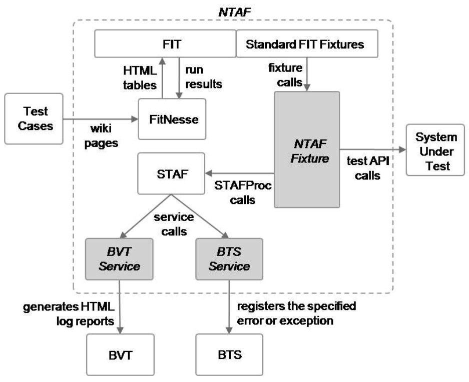


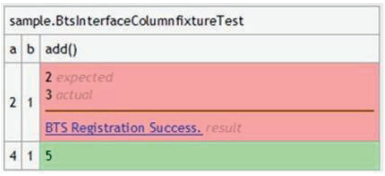
Fig. 1. The NTAF Architecture

2.1 Configurability

NTAF enables an end-to-end solution for test automation. The component of NTAF, NTAF Fixture, provides control structures to handle the execution flow of the job. The NTAF Fixture has a simple but powerful syntax which controls the flow of execution by using commands that redirect the path of execution when appropriate. Originally, Fit tables are executed in sequence. However, NTAF Fixture is designed to make it significantly easier to automate the workflow of tests and test environments. They can represent loops to execute a task repeatedly (e.g. LOOP, ITERATE, CONTINUE, BREAK), parallel executions to be executed in parallel (e.g. PARALLEL, PARALLELITERATE), time-based executions to control the length of execution time (e.g. TIMER), logic flows which evaluate conditions (e.g. IF/ ELSEIF/ ELSE), and the keywords may be nested [2]. They support distributed environments so that multiple servers with clients can be configured and tested at the same time. It is very effective in reducing duplicate codes and modules from test applications.

2.2 Reusability

We developed our own services to interface with Build Verification Test (BVT) and Bug Tracking System (BTS), reusable components that provide all the capability in STAF. Each service is provided in a jar file. The BVT service is invoked from the STAF service request to build a test suite. Before submitting a STAF service request, the BVT service must be registered via the configuration file [2]. STAF service request can be sent to the BVT service on the local machine to another or remote machine in the STAF environment. BVTs are usually run on a set schedule automatically. A build is considered a success if all the tests in the BVT have passed. After building a test suite, FitNesse results page with an unexpected value marked in red is displayed as shown in Fig. 2. When errors or exceptions arise in testing actions, the BTS service registers them automatically so that BTS can help developers and testers keep track of reported software bugs in the work.

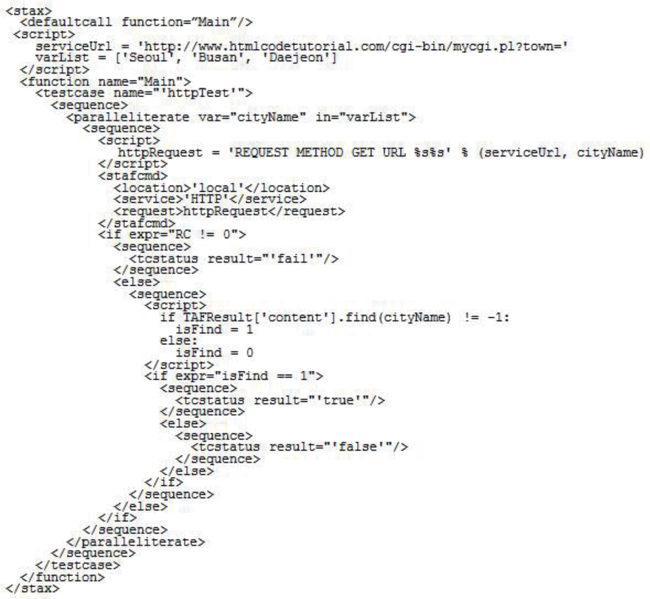


sample.BtsInterfaceColumnfixtureTest		
a	b	add()
2	1	2 *expected* 3 *actual* BTS Registration Success. *result*
4	1	5

Fig. 2. Automated bug report to the BTS

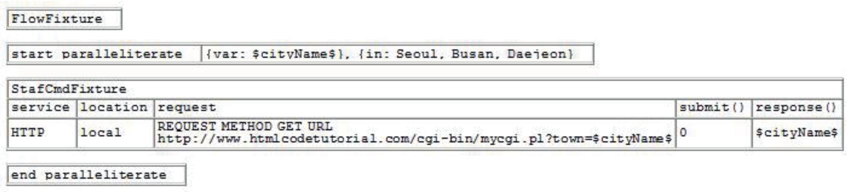
2.3 Visibility

Fig. 3 shows an example of the STAX job for testing web application. It is for running the HTTP service in multi-thread environment. However, writing STAX xml job file is not easy and the workflow of test is difficult to understand through xml file. To write tests and check the results in the tabular format is very simple and clear to the various stakeholders and developers as shown in Fig. 4. Fig. 4 is the same test workflow as Fig. 3. It is available for a developer or tester to write tests and configure the workflow of tests by tables. NTAF takes advantage of the powerful syntax of STAF to be able to develop better Fit tests. It reduces the effort for creating workflows of test cases and increases the visibility of test workflow. NTAF is suitable for running regression test suites to compare the results of the same tests run against a previous version of tables.



```
<stax>
  <defaultcall function="Main"/>
 <script>
    serviceUrl = 'http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl?town='
    varList = ['Seoul', 'Busan', 'Daejeon']
  </script>
  <function name="Main">
    <testcase name="'httpTest'">
      <sequence>
        <paralleliterate var="cityName" in="varList">
          <sequence>
            <script>
              httpRequest = 'REQUEST METHOD GET URL %s%s' % (serviceUrl, cityName)
            </script>
            <stafcmd>
              <location>'local'</location>
              <service>'HTTP'</service>
              <request>httpRequest</request>
            </stafcmd>
            <if expr="RC != 0">
              <sequence>
                <tcstatus result="'fail'"/>
              </sequence>
              <else>
                <sequence>
                  <script>
                    if TAFResult['content'].find(cityName) != -1:
                      isFind = 1
                    else:
                      isFind = 0
                  </script>
                  <if expr="isFind == 1">
                    <sequence>
                      <tcstatus result="'true'"/>
                    </sequence>
                    <else>
                      <sequence>
                        <tcstatus result="'false'"/>
                      </sequence>
                    </else>
                  </if>
                </sequence>
              </else>
            </if>
          </sequence>
        </paralleliterate>
      </sequence>
    </testcase>
  </function>
</stax>
```

Fig. 3. Example of the STAX job definition



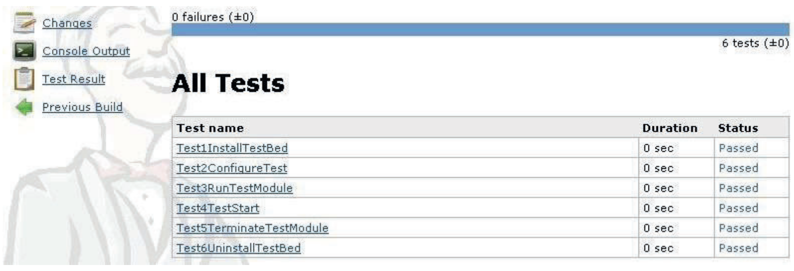
FlowFixture

start paralleliterate	{var: $cityName$}, {in: Seoul, Busan, Daejeon}

StafCmdFixture				
service	location	request	submit()	response()
HTTP	local	REQUEST METHOD GET URL http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl?town=$cityName$	0	$cityName$

end paralleliterate

Fig. 4. Example of the NTAF workflow



Changes
Console Output
Test Result
Previous Build

0 failures (±0)

6 tests (±0)

All Tests

Test name	Duration	Status
Test1InstallTestBed	0 sec	Passed
Test2ConfigureTest	0 sec	Passed
Test3RunTestModule	0 sec	Passed
Test4TestStart	0 sec	Passed
Test5TerminateTestModule	0 sec	Passed
Test6UninstallTestBed	0 sec	Passed

Fig. 5. Hudson Dashboard Displaying the Latest Build Status

2.4 Extensibility

NTAF provides powerful compatibility and extensibility. We selected the Hudson and incorporated NTAF into it. In previous work [3] we incorporated NTAF into the CruiseControl to perform a CI environment. It could be alternated with other CI servers or build tools that can poll for changes in the version control repository on a specified time interval [4]. Fig. 5 illustrates Hudson dashboard to visualize our project status. It shows inspection reports, code metrics, and the results of the last build and view build reports, including compilation errors, automated test results, and details about what files have changed since the last build of our project.

3 Conclusion

We have been using NTAF for testing the practical products, such as database management tool, web application, network library, open API, enterprise service bus system, HTTP service, and XSS (Cross Site Scripting) filtering service at the NHN Corporation. Our case studies indicate that the use of NTAF can aid teams in developing a higher quality product.

Concurrent defects are often hard to find in the testing environment and are therefore found by the end user, or in stress test, which makes them very expensive [5]. Therefore, we have started to work on integrating with a tool for testing multi-threaded programs on NTAF. Concurrent faults may be automatically detected by a tool that exposes and eliminates concurrency-related bugs if NTAF executes each test multiple times. In the future, NTAF can be used as a full-featured, cost-effective automation framework including an effective method for finding concurrent defects.

Finally, we want to emphasize that NTAF is an open source framework. We recommend you to download NTAF and try it out. You can contact us if you would like to participate in the development process.

References

1. Mugrigde, R., Cunningham, W.: Fit for Developing Software. Prentice Hall, Englewood Cliffs (2005)
2. Software Testing Automation Framework (STAF), `http://staf.sourceforge.net`
3. Kim, E., Na, J., Ryoo, S.: Test Automation Framework for Implementing Continuous Integration. In: 6th International Conference on Information Technology: New Generations. IEEE Press, Los Alamitos (2009)
4. Duvall, P.M., Matyas, S., Glover, A.: Continuous Integration. Addison-Wesley, Boston (2007)
5. Edelstein, O., Farchi, E., Goldin, E., Nir, Y., Ratsaby, G., Ur, S.: Testing Multi-threaded Java Programs. IBM System Journal Special Issue on Software Testing 41(1) (Feburary 2002)

Long-Term Effects of Test-Driven Development A Case Study

Artem Marchenko[1], Pekka Abrahamsson[2], and Tuomas Ihme[3]

[1] Nokia, Visiokatu 3, FIN-33720 Tampere, Finland
artem.marchenko@nokia.com
[2] Deparment of Computer Science, P.O.Box 68, FIN-00014 University of Helsinki, Finland
pekka.abrahamsson@cs.helsinki.fi
[3] VTT Technical Research Centre of Finland, P.O.Box 1100, FIN-90571 Oulu, Finland
tuomas.ihme@vtt.fi

Abstract. Test-Driven Development (TDD) is one of the most widely debated agile practices. There are a number of claims about its effect on the software quality and team productivity. The current studies present contradicting results and very little research has been performed with industrial projects, which have used TDD over an extensive period of time. This paper is reporting the long-term effects on a three year-long application of TDD in a Nokia Siemens Networks team. We present qualitative findings based on interviews with the team members. We conclude that TDD has been found to improve the team confidence in the code quality and simplify significantly the software maintenance. The examined team did not notice any significant negative effects over the long-term TDD application and is eager to continue improving the practice application. The authors suggest that results bear direct relevance to the industry and academia. Further research avenues are indicated.

Keywords: Test-Driven Development, agile, case study, long-term.

1 Introduction

Lately agile software development methods have gained a significant amount of attention. Test Driven Development (TDD) is the core part of the contemporary agile practice. It is claimed to raise significantly the code quality and handle the requirements better, than "more traditional, heavy-weight, predictive methods"[1]. There are not many studies on the long-term effects of TDD especially in the industrial context. Existing studies in the industrial context present controversial results varying from none to significant improvements in quality and small improvements to reasonable losses in productivity.

This paper aims at filling in this knowledge gap and addresses the following research question: *What are the effects of the long-term application of the Test Driven Development?*

In the current paper we report on the longitudinal application of TDD by a team working for Nokia Siemens Networks in Finland. With the help of semi-structured interviews we gathered the team experiences and perceptions on what it means for them to apply TDD and what effects of TDD they experienced.

The rest of the paper is organized as follows: section 2 provides an overview of prior research on the effects of Test-Driven Development. Section 3 describes the research environment, method and setting. Section 4 presents the results. Section 5 presents the discussion on the findings and section 6 concludes.

2 Related Work

Lately agile methods gained popularity increasingly. It is evident that at the moment there are not many empirical studies on it. The most recent and in-depth systematic review of 1996 articles and studies in the area of agile software development identified only 36 empirical studies of acceptable rigor, credibility and relevance [2]. However, there are a number of studies on the Test-Driven Development technique. For the purposes of this study we reviewed only studies performed in the industrial context, i.e. with professional developers working on the applications demanded by business.

Majority of the studies report results on short-term projects. Only Williams et al. [3] studied a project in which TDD was applied for longer than a year.

2.1 Studies Conducted in an Industrial Context

Nagappan et al. [4] report on three Microsoft and one IBM studies that pre-release defect density of the four products decreased between 40% and 90% relative to similar projects that did not use the TDD practice. Subjectively, the teams experienced a 15-35% increase in initial development time after adopting TDD.

Maximillen and Williams [5] assessed Test-Driven Development at IBM on a project of 71 KLOC of non-test code and found that the application of TDD reduced the defect rate by about 50% compared to a similar system that was built using an ad-hoc unit testing. The study also found a slight decrease in developer productivity when employing the TDD practice. Maximillen and Williams also suggest that the TDD practice aided them in producing a product that would more easily incorporate late changes.

Bhat and Nagappan [6] studied two Microsoft teams that practiced TDD on a medium sized projects (6 KLOC and 24 man-months, 26 KLOC and 46 man-months) and compared them to the similar teams doing similar projects without TDD. The results of the study indicate significant increase in quality of the code (greater, than two times) for projects developed using TDD. However, the results also indicate that the projects also have taken at least 15% extra upfront time for writing the tests although the unit tests have also served as auto-documentation for the code as well as for code maintenance.

Williams et al. [3] studied an IBM team who had sustained the use of TDD for five years and over ten product releases. Their data indicates that the TDD practice can aid in the production of high quality products. The study notes the perceived moderate productivity losses, but suggests that the quality improvement will compensate for it. In particular the authors suggest that the use of TDD may decrease the degree to which code complexity increases as software ages.

Williams et al. [7] studied the development of a relatively large (64 KLOC of new code) product at IBM and found out that the code developed using TDD showed approximately 40% fewer defects during functional verification and regression tests compared to the similar project developed without TDD. The study claims that the result has been achieved with "minimal" impact to developer productivity.

Rendell [8] examined the introduction of TDD to a large project and identified that even after initial enthusiasm developers could come to resent TDD as they perceive it as a technique which reduces their efficiency. Rendell also identified that dogmatic TDD application could lead to increased costs without the corresponding return on investment. Rendell also notes that for more experienced developers the use of TDD helps to make the code more testable.

3 Research Design

This study covers the results of adopting Test-Driven Development and applying it for three years in a particular team working for a global telecommunications company.

3.1 Research Setting

The examined team works for Nokia Siemens Networks on a Java-based rich client platform for network management-related applications.

In 2005, the platform originally developed in Germany was moved to Finland and since then the whole team, except for a single expert consultant, was co-located in the city of Espoo. The team is following the Scrum process [9]. At the moment there are five developers, a lead designer, Scrum Master and Product Owner. The team is a part of a bigger program that is located partially in the same building, partially in other cities.

At the moment of transfer to Finland according to the lead designer impressions the platform consisted of legacy code [11] with a significant amount of bugs, no more than 10-20 unit tests "not done in TDD way and the tests were really terrible". According to another team member, no tests were made at all and the first poor tests were made by the current team, not yet proficient in TDD.

The adoption of Test-Driven Development was happening not in isolation, but concurrently with the adoption of several related agile practices. In particular, at about the same time the team improved the Scrum process introduced shortly before by starting to apply pair programming, collective code ownership and continuous integration.

In summer 2005, shortly after the move to Espoo two of the team members including the lead designer participated in a TDD training and the lead designer was "test-infected". A program-level decision was made to raise the test coverage of the code (it was about 9% at that moment) and the software developers started writing tests primarily with the goal of raising test coverage, mostly during separate workshops.

Around the same time the lead designer participated in a one week "hands-on" training run by a TDD coach, which provided him with a practical experience of working in a TDD manner with the team's own code. A number of other issues were also happening simultaneously. Refactoring was almost non-existent before TDD, and pair programming was adopted as a natural part of TDD with one of the persons "navigating" the design towards the desired state.

After two years into the adoption of TDD an expert from the same consulting company was playing the role of Scrum Master and was driving the TDD efforts. More recently, he was replaced with an in-house Scrum Master.

3.2 Research Method

The study was conducted using the case study research method [10]. It was seen as the most appropriate choice due the nature and aims of the study. The development team was contacted in October 2008. In November the lead designer was interviewed over teleconference with notes shared via a screen sharing program so that the interviewee was able to correct the notes during the interview.

During December all the current team members, including the lead designer and one of their indirect managers (site manager) who was working with the team for 19 months, were interviewed in-person individually with a total interview record time of 4 hours and 40 minutes. Semi-structured interviews were chosen in order to get the interviewees' perception of the area. Questions prepared in advance were used as discussion openers. This was also used by the interviewer to make sure a particular topic would not be missed. Each time a change was mentioned during the discussion the time of the change was asked by the interviewer. The interview results were then systematically interpreted by comparing the interview notes looking for common trends and differences. The detailed list of questions is omitted here due to the lack of space. During the interviews we aimed at figuring out:

1. What it meant for the team to apply TDD
2. How the TDD practice was evolving
3. What effect the team saw from applying TDD

3.3 Data Collection and Analysis

During the research we arranged 9 interviews with 8 people, adding to 4 hours and 40 minutes of records outlined in Table 1. The two interviews were particularly short because the people interviewed either joined the team activities only

Table 1. Amount of interview notes

Person	Time on the team	Interview length	Notes length
Lead designer	Three years	68 and 9 min	1206 and 174 words
Software developer	Almost a year	57 min	798 words
Software developer	A bit more, than a year	28 min	707 words
Site manager	19 months	43 min	Formal notes were not taken, only audio records
Software developer	One month	8 min	207 words
Software developer	Three-five years	32 min	1055 words
Software developer / UI Designer	Eight months 26 min	399 words	
Line manager	One year	13 min	241 words

lately or they were not involved in the development activities and therefore were seen as the type of informants that would not contribute to the study's aim. As we were looking for the team's own observation about the effects, we were using the site manager's interview only for providing a context for the preparation to the main interview session.

The interviews were analyzed by systematically extracting statements and claims from the interviews together with a related time range whenever possible. A sample extraction from a statement list is as follows:

– The consultant taught them to do TDD for every line of code (*2 years ago*), but in reality he does not do it for every line of code.
– "In some cases if I know where this (the code) is going (when using some design pattern) then TDD is not used for every line of code.
– "I have learned some of my own understanding of implementation of TDD (*during the last year*). The main point is to test first and then (create) the implementation."
– Has learned more unit testing (jUnit), how to test user interfaces using unit test utilities (UI for J or some extension for jUnit). (*during the last year*)

Finally, to validate our findings we used the member-checking approach and asked all team members to study the validity of our interpretation. This subsequently led to increased understanding and some minor modifications in our results.

4 Empirical Results

In this section we present the aggregated summary of the interviews along with the perceived challenges related to TDD. All the citations listed in this section are quotes of the interviewees, sometimes slightly edited for readability.

4.1 What It Meant for the Team to Apply TDD

Most of the interviewees described TDD exactly as it was described in the appropriate literature [12] - "You do not modify your code if you don't have a red test that will turn green after your modification." Interestingly only one team member mentioned refactoring as a TDD step when describing the TDD practice. However, when discussion was on-going, all the team members clearly described the importance of the refactoring step in the Test-Driven Development cycle up to the point of claiming that the ease of refactoring was one of the main advantages of the TDD technique - "TDD gives you a possibility to refactor your code," "You have a pretty high safety in modifying your code," "One of the biggest effects [of TDD] is that it is easy to do even big changes in the code as long as it does the same thing."

Two team members mentioned that refactoring was there to remove the duplication from the code. One team members explicitly mentioned the need to refactor the test code as well.

Interestingly all the team members, except for one newcomer, explicitly mentioned pair programming as a necessary and natural part of TDD. Further discussion revealed that at the moment all the production code of the team was developed in the pair programming mode. "I often confuse these two things. I often think that TDD is pair programming."

4.2 Evolution of TDD Since the Adoption

To our surprise, the team members identified almost no practice evolution over the long time period claiming that most of the changes related to TDD are in the psychological area - from the beginning it was very hard to write tests first for all the production code - "TDD is slower in the beginning, but it produces much more healthy code". Several team members also mentioned that refactoring skills evolved over time as practitioners learned how to test-drive the code to a known or standard (for the team) patterns [13].

4.3 Perceived Positive Effect of TDD

The interviewees identified the following positive effects of TDD:

Improved code quality and high level of confidence in the code quality. All the team members identified the increased confidence in the code quality as one of the main effects of TDD. All the team members felt that TDD improved the code quality and made the defect fixing easier. "I found TDD useful in many cases, especially for bug fixes," "TDD reduces a number of bugs. I would say it improves my way of working 10 to 20 per cent," "The difference is that now we know that the code is working as opposed to the time when we did not have an idea if the code was working or not, when three years ago we did not have tests."

Improved code readability, understanding of the code and ability to make even significant changes easily. All the team members noted that TDD made the individual class code more readable. One of the team members told that after he had started to apply TDD, it had helped him realize the reasons for poor maintainability of the code he had used to write earlier. "We are talking about cohesion and dependencies, how much we have dependencies in a class. TDD revealed those kinds of things in my code. I guess I understood the reasons why I have created code that was not so well maintainable." The team members also noted that TDD not only helps to reveal the bad code, but also helps to improve it even when significant changes are needed - "One of the biggest effects is that it is easy to do even big changes in the code as long as it does the same thing."

Having less overhead in code and documentation. Only the amount of code and documentation that was really needed was created. "Now I am writing code that is really needed, because I won't do any production code if I don't have a test. So I am not putting any extra stuff into my code anymore," "Test-driven means that you specify the code by writing tests. You don't write a document like you have done waterfall. There isn't that much documentation anymore, all the documents are in the tests. In the test you define how your software should work. I haven't written any documents since I joined the team."

4.4 Perceived Challenges in Applying TDD

All the team members were asked about the perceived challenges related to the application of Test-Driven development. The challenges identified by the interviewees were:

The high amount of discipline needed to rigorously apply TDD. Rigorous application of TDD requires an amount of discipline that can be rather high for a team new to it. "Sometimes it is so evident what corrections should be made. The only problem is that there can be side-effects", "The most challenging thing is to remember to use TDD."

Applying TDD to the graphical user interface development. TDDing user interfaces is more difficult because of both the technical complexity and qualitative (human perception related) nature of some requirements - "In UI design you design the UI first without making any code. It may have been already tested (paper, prototype) before making any code". UI designer was the only team member that reported that he was not doing all the code via TDD. "In April I used TDD 0% of time, now 50%, the amount of time spent with TDD has increased at an increasing rate." "At the moment, however, some team members always develop new GUI with TDD."

Applying TDD to configuration related areas. "Very hard to invent how to do TDD when changes are made at XML files."

Applying TDD to legacy code that does not have tests yet. It takes time to implement the very first tests in the given code area - "You need to study this legacy code first".

4.5 Perceived Negative Effects Related to TDD

Negative effects of the TDD practice identified by the interviewees were:

Inefficiency for fixing defects that are difficult to reproduce. TDD is difficult to use for the bugs that are difficult to reproduce and for bugs that need a very complex and specific environment - "It doesn't make any sense to automate it. It would be cumbersome."

Potential decrease of the development speed. TDD can slow down the development speed. "When you start it can easily slow you down ten times. 15 minutes' hack might take one day with TDD. Outcome gives better maintainability, but sometimes it is too expensive." On the other hand another team member "doesn't feel that TDD has affected productivity."

Too open and flat architecture. One of the team members told that TDD encouraged the creation of many small files that, in his opinion, made the code less readable than smaller amount of bigger files - "From my traditional coding point of view a bit bigger pieces and less of them is easier to read." Another team member told that the only thing that bothers him in TDD was that it made the design more open - "Sometimes from the design point of view you have to open the class more than you would like to. To be testable you need to reveal more internals of the class. That's one thing, that I don't like, but so far I haven't seen any bad effects of that."

5 Discussion

This study reports on the perceptions of the team that was rigorously applying the Test-Driven Development for three years. This study identified several positive and negative effects of the long-term TDD application. In particular it was discovered that after years of applying TDD it could still be considered a very valuable practice the team would not like to stop using.

The code quality-related results that confirm the increased level of the code quality are in line with all the other reviewed studies except for the study focused on the TDD introduction only [8]. Our study reports mixed impressions about the development speed. Some team members claim that TDD slows them down, while others tell that TDD makes them more productive. All the team members, however, are confident that over the long term TDD pays for itself. That is in line with the related studies and especially with the study on the five year project performed by Williams et al. [3] that also suggests that the use of TDD may decrease the degree to which code complexity increases as software ages. Further research is needed for figuring out the "payoff point" and related factors.

During the interviews, most of the team members related the success of the TDD adoption to the excellent initial trainings that included practical work on the team's own code and to the year-long presence of external TDD experts in the team who helped the team learn the agile way. Further research is needed for analyzing how much TDD success depends on the quality of training and coaching.

The study also identified several effects of TDD that were not previously identified in the related literature. These effects include a potential tendency for the development of an architecture that is both too open and flat. We also discovered that some of the team members perceived that TDD covers most of the code level documentation needs. This is in accordance with what the proponents of TDD have claimed for some time. The study did not reveal any negative impacts for the team on having adopted this practice. Quite contrary, they viewed this as a positive impact of TDD. Further research is needed for figuring out whether such effects are common or specific to the particular team.

Summarizing the findings, we suggest that Test-Driven Development is a viable technique for multi-year projects that may very well pay off for the potential productivity decreases and lead to higher quality products that are easier to change according to the dynamic market requirements.

There are important limitations on the study. The team studied did not adopt the TDD practice only, instead it came together with pair programming and Scrum-based management practices. It has to be researched whether the combination of the practices was the real success factor. The team studied was using Java which is known for good tooling support for unit-testing and refactoring. The Java team results might be less applicable to the environments with poorer tooling support, such as embedded C++. Finally, we investigated only a single team's adoption perceptions. This should be taken into account when attempting to generalize the results.

The study results are qualitative and based on the interview results and a member-checking validation of these results. Clearly, further research is needed for analyzing the measurable quantitative effects of TDD. Further research is also welcome on even longer term projects including more teams and focused studies on the effect on the architecture.

6 Conclusion

This research studies the impressions of a team that has applied Test-Driven Development on a rigorous basis for the duration of three years. This study identified several perceived effects of the long term TDD application. We answered the research question with a set of perceived positive and negative effects observed by the team as well as with a set of challenges related to the TDD application.

The results of the observations add to the empirical body of evidence on TDD and bear significance for both industry and academia. Industry has more knowledge on what can be expected from the long-term TDD application and whether it is worth pursuing, academia gets the knowledge gap filled in and new research venues identified.

References

1. Wasmus, H., Gross, H.-G.: Evaluation of Test-Driven Development (2007)
2. Dybå, T., Dingsøyr, T.: Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology (2008)
3. Williams, L., Sanchez, J.C., Maximilien, E.M.: A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry. In: Agile 2007 (2007)
4. Nagappan, N., Maximilien, E., Bhat, T., Williams, L.: Realizing quality improvement through test driven development: results and experiences of four industrial teams. Empirical Software Engineering 13, 289–302 (2008)
5. Maximilien, E.M., Williams, L.: Assessing test-driven development at IBM. In: Proceedings of 25th International Conference on Software Engineering, pp. 564–569 (2003)
6. Bhat, T., Nagappan, N.: Evaluating the efficacy of test-driven development: industrial case studies. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM, Rio de Janeiro (2006)
7. Williams, L., Maximilien, E.M., Vouk, M.: Test-driven development as a defect-reduction practice. In: 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003, pp. 34–45 (2003)
8. Rendell, A.: Effective and Pragmatic Test Driven Development. In: AGILE 2008 Conference, pp. 298–303 (2008)
9. Schwaber, K.: Agile project management with Scrum. Microsoft Press, Redmond (2004)
10. Yin, R.: Case study research: Design and methods, 2nd edn. Sage Publishing, Beverly Hills (1994)
11. Feathers, M.: Working Effectively with Legacy Code. Prentice Hall PTR, Englewood Cliffs (2004)
12. Beck, K.: Extreme programming eXplained: embrace change. Addison-Wesley, Reading (2000)
13. Gamma, E., Helm, R., Johnson, R., John, V.: Design patterns CD elements of reusable object-oriented software. Addison-Wesley, Reading (1998)

Communicating Domain Knowledge in Executable Acceptance Test Driven Development

Shelly Park and Frank Maurer

University of Calgary, Department of Computer Science
2500 University Drive NW, Calgary, Alberta, Canada, T2N 1N4
{sshpark,fmaurer}@ucalgary.ca

Abstract. We present results of a case study looking at how domain knowledge is communicated to developers using executable acceptance test driven development at a large software development company. We collected and analyzed qualitative data on a large software development team's testing practices and their use of a custom-built executable acceptance testing tool. Our findings suggest that executable acceptance tests (1) helps communicate domain knowledge required to write software and (2) can help software developers to communicate the status of the software implementation better. In addition to presenting these findings, we discuss several human aspects involved in facilitating executable acceptance test driven development.

Keywords: Executable Acceptance Test Driven Development, Requirements, Ubiquitous language, Domain knowledge, Knowledge management.

1 Introduction

The purpose of the Executable Acceptance Test Driven Development (EATDD) is to facilitate better communication between the customers, domain experts and the development team by mandating that the requirements must be specified in form of executable acceptance tests. This general idea is currently called many names: functional tests [1], customer tests [2], specification by example [3] and scenario tests [4] among many. Recently, EATDD is becoming very popular in the agile software engineering community.

What is distinct about the EATDD process is that requirements are specified in executable acceptance tests by the customer instead of using natural language. An executable test either succeeds or fails, i.e. there is no ambiguity. Any specification must be understandable and writable by domain experts as well as the development team members. Evans stated that finding a common communication language or a Ubiquitous Language [5] is important in communicating the correct requirements between customers and the developers.

To pursue a better understanding of the functional requirements specification process in EATDD in terms of how the domain knowledge is communicated across different stakeholders, we performed a detailed case study at CGI [6], a large international information technology and business process service firm with offices located in

16 countries. The software developers at CGI in Calgary office were given a challenge to build oil & gas production accounting software by their clients. Production accounting deals with accounting involved in acquisition, exploration, development and production of oil and gas reserves. The production accounting domain knowledge is very complex and no single developer can understand the client's domain in its entirety. The development team follows Agile development practices and started to use executable acceptance test driven development (EATDD) based on its perceived usefulness for communicating requirements between the developers and the business representatives. The team built a custom in-house testing tool to facilitate their requirements elicitation and acceptance testing process. The purpose of our case study is to understand how the custom testing tool helped establish a Ubiquitous language between the business domain experts and the developers. This is a qualitative research paper that looks at how the tools facilitated the communication of complex knowledge to the developers.

The primary audience for our research paper is researchers in EATDD field in Agile development environments. In addition, the broader audiences include industry practitioners who are interested in using EATDD. This paper provides a first in-depth longitudinal case study on the EATDD practice. The project under study is running for four years. We collected data about the software development project and corroborated our findings with interviews and direct observations. We investigated their in-house custom executable acceptance testing tool and its impact on the development *process.*

Section 2 presents related research on EATDD. Section 3 presents a research methodology and our research design. Section 4 provides background information about the development project. Section 5 presents our observation. Section 6 discusses the significance of our research and section 7 presents the threats to validity in our research design. Section 8 concludes the paper with some final thoughts on our case study and presents the next step in our research.

2 Related Work

EATDD is inspired by test-driven development (TDD) [1]. Similar to TDD, EATDD asks for requirements as a set of acceptance tests that can test for functional software requirements. What is distinct about this process is that requirements are specified in executable acceptance testing form by the customer instead of in a natural language. The specification must be writable and readable by customers and the development team members. The practice became particularly popular in the Agile development community, especially with the introduction of Cunningham's testing framework called Fit [7] and their subsequent add-ons such as Fitlibrary[8] and Fitnesse [9]. Similar testing frameworks are also becoming available in the market, e.g. Greenpepper [10]

Melnik et al. conducted a study on how students perceived and used Fit testing framework for specifying software requirements [11]. Their result shows that despite the students' objection to writing the executable acceptance tests in the beginning, all of the students were able to convey the functional requirements completely. At the end of the course, 80% of the students said they didn't want to specify all of the requirements in prose and would prefer a tool like Fit. Read et al. found that about 2/3 of the students said they will use Fit for acceptance tests in the future [12]. Sauvé et al. also corroborated the finding that students were able to improve their communication through their custom testing tool called EasyAccept [13].

Melnik et al. also conducted a study with industry practitioners. They found that customers and testers "recognized the effectiveness of the executable acceptance test-driven development for specifying and communicating functional business requirements", but the developers preferred unit tests over Fit tests. Ricca et al. found that Fit tables can help clarify a set of change requirements in the software with improved correctness with no significant impact on time. However, they found that the benefits of Fit vary by developers' experience [14].

While these studies suggest that executable acceptance tests can improve communication about functional software requirements, the empirical basis is still limited. The existing research work still doesn't explain why some people are very enthusiastic about it and some are very hesitant. We also need to find out how executable acceptance tests should be specified and what additional purpose these tests serve.

3 Research Approach

In this section, we describe our research methodology and our research design. Human factors are difficult to measure because it is impossible to measure the software engineering process independently of the practitioners. When we investigate a real-life software development project, we can't always measure, control or identify all of the factors that influence the development process [15]. Therefore, to facilitate our qualitative research in such settings, we decided to leverage a case study research strategy, which is a set of research procedures that are designed to facilitate research where researchers are interested in "how" and "why" of the phenomena under study but has very little control over the behavioral events. The purpose of using a case study method is to deliberately uncover the contextual conditions that led to the phenomenon "believing that they might be highly pertinent to [the] phenomenon" under study. In a software engineering context, Fenton et al. [16] label case studies as "research into the typical". Case studies can help discover the relationship between humans and the environmental context.

The goal of the case study research strategy is to contribute a hypothesis about the phenomenon under study mainly through *analytical generalization* rather than *statistical generalization* [17]. It means that although we cannot make any definite conclusion based on one case study, a single case study can provide very rich insights into the problem under study, especially when the evidence is consistent with previously existing research work.

Our case study is done at CGI [6]. While we were collaborating with CGI for an extended period of time [18], the fieldwork for the current study was done over two additional days during which we conducted in depth interviews with members of the CGI team. Our results are based on newly collected data as well as insights from previous collaboration. The new data represents over four years of development practice. We interviewed three additional software developers and one project manager for detailed data collection. In addition, we interacted with five additional software developers to corroborate our findings. We did not record the interviews, because recording the interviews was seen as to be too intrusive in the company environment. However, any interesting remarks made by the developers were carefully written down during the interviews along with any observation we made. The length of the interview varied greatly depending on how much information the developers were providing for our research. We also participated in one daily scrum meeting and observed three other daily scrum meetings. Our empirical data also includes direct observation of a developer who was engaged in a debugging process of a failing executable acceptance test, which lasted about one hour. We wanted to understand how acceptance tests were used to fix the failing code.

Our analysis of the collected data involved open coding and code categorization using our field notes [19]. During open coding, we identified a set of codes that could provide most insights into the data from our field notes. Then we categorized the codes to determine the relationships among the identified codes and a list of themes was generated.

4 The PAS Project

A case study assumes that contextual condition is important in the phenomenon under study. Therefore, in this section, we are going to briefly describe the PAS development project. PAS is production accounting software for the petroleum industry. The purpose of oil and gas production accounting software is to keep track of oil and gas production and calculate various capital interests invested in the oil and gas wells. CGI already had an existing production accounting software called CGI Triangle, but the system needed to be rewritten due to the obsolescence of technologies used for its development.

The CGI PAS project was sponsored by four of the largest oil and gas companies in Canada. An oil and gas production accounting system is an extremely critical software system for oil and gas exploration companies, because engineers and production accountants not only have to keep track of their oil and gas productions, but they need to be able to calculate tax and various interests[1] being represented in their oil and gas wells. Because each country and province has their own unique set of regulations on tax and interest calculations, it is important for the oil and gas companies to use software that reflects the regulations for the political jurisdiction where the well exists. The amount of engineering and accounting knowledge involved in the petroleum industry in order to build PAS is so complex that it is absolutely impossible to build the software without having someone with the expertise who can lay out the information properly to the software developers. While all software development requires business domain experts, the problem with PAS is that production accountants need years of training before they understand the domain. The knowledge is not easy to be picked up by developers on the side. The team also needs someone who can keep track of the changing set of government regulations in the oil and gas industry.

The number of software developers in PAS project fluctuated over the last four years of its development; therefore, the amount of knowledge about the project development within the team fluctuated. At the time of the fieldwork, PAS is already deployed and operational in the client's work environment. Our contacts in the company told us that there are about 80 software developers, testers and clerks at the time of the fieldwork. For an Agile development team, it is quite a large team. The team is split up into several subteams (one for each major component).

[1] Interest: how costs and revenues are shared by stakeholders.

The development area is a large open space. Each subteam had, what they called, a SPOC (Single point of contact) and an SME (subject matter expert). A SPOC communicates the progress of the team and addresses any concerns that other teams might have on the component they are building. The SME is the person who has the domain expertise to define the requirements, answer any domain-related questions from the developers and test the end products to ensure that the requirements were correctly understood and implemented by the developers.

Each subteam had 2 to 8 developers and they stay as a team only for the duration of building the specific business component. Each team holds a daily scrum meeting in the morning. Due to the size of the team, each team had separate scrum meetings and each team sent a team representative to inter-team daily scrum meetings. In addition, they kept track of bug lists using Jira [20]. There were 927 'GUI Smoketests' that test the user interface layer, 93 report regression tests and numerous unit tests and other types of tests that we did not look into carefully for this research.

5 Observation

In this section, we present our observations by describing CGI's executable acceptance test-driven development process. We first describe our observation and then we summarize the implications of our findings.

5.1 Choose the Requirements Specification Tool from the Customer's Domain

The requirements specification is defined using the language and formats of the business domain. This can reduce the extra overhead of learning the specification tool by the domain experts (customer representatives). Our case study shows that the domain experts chose Microsoft Excel as their requirements specification tool, which is a standard tool for communicating production accounting data in the oil and gas industry. The standards for the data formatting are regulated by the provincial energy regulatory boards [21] and production accountants generally use Microsoft Excel to facilitate their business communication. Although production accountants are not required to use Microsoft Excel, it is a common practice to do so in the oil and gas industry.

Microsoft Excel became a preferred tool for requirements specification, because Excel was familiar to the domain experts. Excel has features such as VB macro programming and pivot tables. The domain experts understood and used these Microsoft Excel features proficiently. An acceptance test file has 12 macros, which are used to create these table templates and help with the test automation process.

One Excel file contained many requirements and the developers considered one Excel file as one acceptance test. Each acceptance test is composed of many calculation worksheets, which are represented using Excel worksheets. Each worksheet can contain 20 to 650 rows of test data and about 3 to 30 columns. Each row can return more than one expected output result.

There are a total of 17 business components in the PAS project. Each business component had multiple acceptance tests and each acceptance tests had multiple worksheets (or the developers called them 'views'). There are a total of 321 acceptance tests for the PAS project. The domain experts were able to write and maintain these tests whenever a feature is added or changed in the software.

The use of Excel was motivated by the ease of transferring and codifying the domain knowledge via this medium by the domain experts, especially because Excel is the conventional tool in the business domain. It is easier to codify tacit knowledge to explicit knowledge if the tool is already utilized to facilitate communication in the customer's domain. In addition, (1) domain experts who are familiar with the specification tool are more likely to create and maintain acceptance tests for the developers and (2) the tool can communicate the domain knowledge best because it can represent the domain data appropriately. What makes Microsoft Excel interesting is that this tool is universally well known and easy to learn by both the customers and developers. Finding a common tool that can be used and understood by business experts as well as the development team is a crucial and important condition for a successful EATDD process. Excel not only allowed the production accountants to leverage their existing computer skills for writing the requirements, but it also provided the contents in a form that developers can easily turn into acceptance tests.

5.2 Communicating the Business Domain Knowledge

In this section, we are going to analyze the kind of knowledge their acceptance tests are conveying to the developers. Due to the page limitation, Table 1 shows only a very small snapshot of a large executable acceptance test that has 644 rows and 14 columns, which is a typical example set of scenarios required to test for real-life production accounting. Table 1 is testing for a contract allocation. The last three columns show the sum of the expected share of the energy for everyone who has a stake in the reserves profit. We found that most of the tests are transaction style calculations. Generally the tests identify *where (a physical entity* such as reserves or a facility), *who* the transaction is for, *what* values should be assigned and optionally *when* the transaction occurs. The format is typical of any production accounting spreadsheets. The test data is created by the domain experts using similar production data, but the data is so realistic that it could be the real production accounting data.

Table 1. An Example Snapshot of Executable Acceptance Test Definition

Fac ID	Cont Name	CParty Name	Prod Code	Cont Type	Settlement	Sum of VOL	Sum of PRICE	Sum of VAL
1000405 00722W 500	FH royalty C5	Farmer Cassie	C5-SP	Royalty	Cash	2.76	339.98	938.37
					Cash	2.76	339.98	938.37
	FH TIK on gas – 100%	Kathy and Co.	GAS	Royalty	TIK	2.2	235.69	518.51
					TIK	2.2	235.69	518.51

The domain experts also provided workflow diagrams that are not executable, but they complement the executable specification. They were designed to inform the developers about the business workflow for the acceptance test. Figure 1 shows a

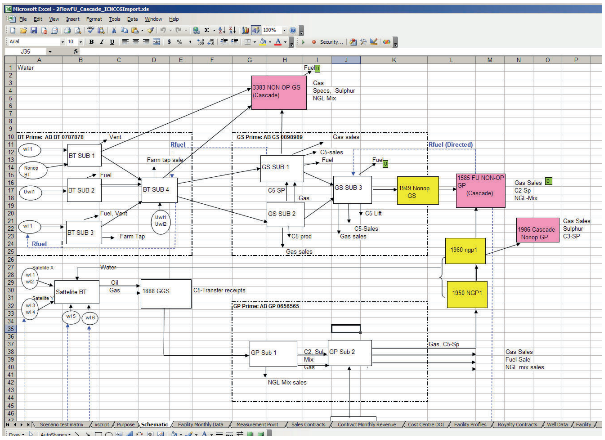


Fig. 1. A diagram explaining the business process involved in a battery facility

workflow diagram of a battery facility[2]. Our analysis shows that the domain experts need to provide two types of documents: testable requirements specifications and an overview document to put the specification into context. The overview document can be used by the developers to point at something to ask for more information about the domain.

5.3 Making the Requirements Specification Executable

The testing framework is based on Excel, Ruby and JUnit. Ruby is used to automate the user interface layer. JUnit is used to execute the script file and to compare the test output values. Executing one acceptance test can take up to 1 hour or more, because it simulates real-life production data and calculations and it runs against the UI layer. A production data file contains months or even years of production data. The largest Excel file is about 32 megabytes in size.

Figure 2 shows a time-series analysis graph of percentages of succeeding acceptance tests captured at the end of each sprint. Notice that over time, the developers became much more conscience about passing the acceptance tests. But we cannot make a definite conclusion about the quality of the software from the graph.

We wanted to get better understanding whether the graph reflected the amount of domain knowledge that was transferred to the developers. We asked the developers to explain specific parts of the acceptance test that were failing. They explained it in terms of how they can fix the problem technically, but they could not explain the domain knowledge behind these tests by putting it into context of the industry. However, they knew enough to explain why the test would fail. Based on our interview data, we can assume that the team's understanding of the business domain is fragmented across many developers. It also meant the executable acceptance tests provide enough information for the developers to implement the code even if they have limited understanding about the domain in which it will be used. The regression tests using executable

[2] Battery facility: a plant where raw petroleum is separated into different types of hydrocarbons.

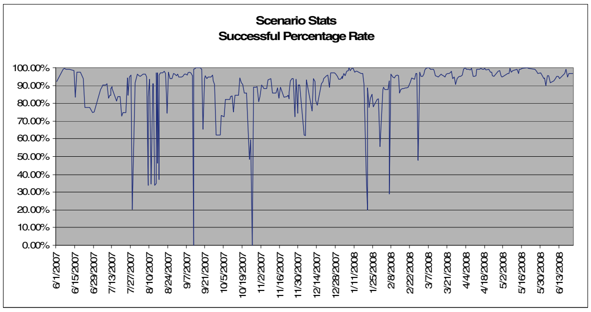


Fig. 2. A time-series graph showing the percentage of acceptance tests succeeding at the end of each sprint. 0% success rate was due to a test automation problem at the time rather than any serious software malfunction.

acceptance tests were important because they highlighted this knowledge gap among the developers. It made the knowledge gap transparent to everyone. Therefore, they knew who and when to seek out help when these tests failed. Unlike unit tests, failure in executable acceptance tests meant they misunderstood domain knowledge, thus it signaled possible problems in delivering business value to the customer.

They did not need in-depth production accounting training to fix the software problem, because executable acceptance tests usually gave enough information to identify the problematic code and also provided the expected answer. However, a more in-depth empirical study is required to understand the developer's cognitive processes involved in going from failing executable acceptance tests to fixing the code.

6 Discussion

We categorize our findings into three categories: *communication, medium* and *context.* We discovered that the purpose of having executable acceptance tests is to *communicate* the domain knowledge to the software developers. The acceptance tests were a feedback system for the developers to confirm their understanding about the requirements and the business domain. Previous research also validates our finding [23, 24]. More than any other types of development artifacts, executable acceptance tests are utilized to fill the knowledge gap between the domain experts and the software developers. The failing tests from the automated regression tests are a good starting point for developers to question their understanding about the domain. Without such feedback system, the developers would not know how to validate their understanding about the requirements. However, no one previously looked at how specific types of domain knowledge are written in executable acceptance tests.

The *medium* used for acceptance test specification is important for successful EATDD process. Previous papers found that tools are important [24]. However, we discovered that tools are important for the domain experts more than the developers. The team discovered that production accounting knowledge is very well organized using Microsoft Excel. We hypothesize that Microsoft Excel made the test specification easier for the domain experts. By giving more power to the domain experts, the developers were able to gain valuable acceptance tests that became critical artifacts for their success. We believe that directly utilizing the language and formalism of the business domain (instead of development oriented languages and formalisms) will improve communication between the business side and the development side of a software project.

We also discovered that the *context* is also important. First, we discovered that the acceptance tests not only provided testable example data, but we also need to provide overview documentation about the domain knowledge and business workflow diagrams. The extra information helped communicate the necessary domain knowledge needed to understand the acceptance test specifications.

7 Threats to Validity

For external validity, we believe our case study can be generalized to very large software development projects where the software developers do not have complete understanding of the domain knowledge. However, our case study only supports transactional style domain data. For threats to internal validity, the observation was collected and analyzed mainly by the first author, which may have introduced some unintended bias. Our interview data also mostly reflect the perspectives of the developers. To ensure construct validity, we performed a detailed inspection of the tool and the requirements. We are confident about our findings, because our observation and test data represents over four years of development practice.

8 Conclusion

Our case study suggests that a successful software development process is one that allows the developers to quickly verify and validate their understanding about the domain knowledge as well as the software requirements. However, in order to do so, the software developers require a reason to believe that their domain understanding needs further inquiry. An automated feedback system, especially using executable acceptance tests, allows the developers to ask the right question to fulfill their task.

References

1. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (1999)
2. Jeffries, R.: What is XP?, `http://xprogramming.com/xpmag/acsFirstAcceptanceTest.htm`
3. Fowler, M.: Specification by Example
4. `http://www.martinfowler.com/bliki/SpecificationByExample.html`
5. Kaner, C.: Cem Kaner on Scenario Testing: The Power of 'What-If' and Nine Ways to Fuel Your Imagination. Better Software 5(5), 16–22 (2003)
6. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, Reading (2003)
7. CGI, `http://www.cgi.com`

8. Fit, http://fit.c2.com
9. FitLibrary, http://sourceforge.net/projects/fitlibrary
10. Fitnesse, http://fitnesse.org
11. Greenpepper, http://www.greenpeppersoftware.com/
12. Melnik, G., Maurer, F.: The Practice of Specifying Requirements Using Executable Acceptance Tests in Computer Science Courses. In: Proc. of 20th OOPSLA 2005, San Diego, USA, October 16-20, 2008 (2005)
13. Read, K., Melnik, G., Maurer, F.: Student Experiences with Executable Acceptance Testing. In: Proc. of Agile Conference 2005, July 24-29, pp. 312–317. IEEE Press, Los Alamitos (2005)
14. Sauvé, J.P., Neto, O.L.: Teaching software development with ATDD and EasyAccept. In: Proc. of SIGCSE tech. symp. on CPSC Education, Portland, USA, pp. 542–546 (2008)
15. Ricca, F., Penta, M., Torchiano, M., Tonella, P., Ceccato, M., Visaggio, C.: Are Fit tables really talking?: A series of experiments to understand whether fit tables are useful during evolution tasks. In: Proc. of the 30th ICSE 2008, Leipzig, Germany, pp. 361–370 (2008)
16. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Dordrecht (2001)
17. Fenton, N., Pfleeger, S.: Software Metrics: A rigorous and practical approach, 2nd edn. PWS Publishing Company (1996)
18. Yin, R.: Case Study Research: Design and Metrics. Sage Publications, Thousand Oaks (2003)
19. Grewal, H., Maurer, F.: Scaling Agile Methodologies for Developing a Production Accounting System for the Oil & Gas Industry. In: Proc. of Agile (2007)
20. Strauss, A., Corbin, J.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Thousand Oaks (1998)
21. Jira, http://www.jira.com/
22. Energy Resources Conservation Board, http://www.ercb.ca/
23. Melnik, G.: Empirical Analysis of Executable Acceptance Test Driven Development, Ph.D Thesis, University of Calgary, Department of Computer Science (August 2007)
24. Melnik, G., Maurer, F., Chiasson, M.: Executable acceptance tests for communicating business requirements: customer perspective. In: Agile 2006, Minneapolis, MN (2006)

An Empirical Study on the TDD Conformance of Novice and Expert Pair Programmers

Andreas Höfer[1] and Marc Philipp[2]

[1] Universität Karlsruhe (TH), IPD Institute
Am Fasanengarten 5, D-76131 Karlsruhe, Germany
Tel.: +49 721 608-7344
ahoefer@ipd.uni-karlsruhe.de
[2] andrena objects ag
Albert-Nestler-Straße 11, D-76131 Karlsruhe, Germany
Tel.: +49 721 6105-126
marc.philipp@andrena.de

Abstract. We conducted a quasi-experiment comparing the conformance to the test-driven development (TDD) process of one expert and two novice groups of programmers working in pairs. Besides an insignificant tendency of the expert group toward a higher TDD conformance and instruction coverage, we found that the expert group had refactored their code to a larger extent than the two novice groups. More surprisingly though, the pairs in the expert group were significantly slower than the pairs in one of the novice groups.

Keywords: test-driven development, pair programming, experts and novices, quasi-experiment.

1 Introduction

Test-driven development (TDD) and pair programming (PP) are agile techniques commonly used in conjunction. Among pair programmers applying TDD in the field we find the whole spectrum in expertise ranging from novices, who have just begun to explore these techniques, to experts, who have been using them for several years in industry. It seems apparent that the level of expertise has an influence on how a programmer pair implements the TDD process and therefore also on the results the pair can achieve. Yet, the actual effects are not easily predictable. Therefore, we conducted a quasi-experiment comparing the TDD processes of one expert and two novice groups of programmers working in pairs. This quasi-experiment yielded rather surprising results which we present in this article.

2 Related Work

In recent years, several studies have been published which compared TDD to other development processes using students as subjects. Müller and Hagner [1] for example, compared TDD to a traditional test-last approach. They found no evidence for differences concerning problem solving time and program reliability but the test-first group made fewer errors when reusing existing methods. Erdogmus et al. [2] conducted an empirical study with computer science students. The test-first programmers wrote more tests per unit of effort but there was no difference between the two groups concerning code quality and productivity. Pančur et al. [3] conducted another study with students. They could not provide evidence for differences between TDD and the test-last development process used. Edwards [4] adopted TDD in a computer science course. His students reduced their defect rate dramatically after adopting TDD and were more confident in the correctness of their programs. Similarly, Kaufmann and Janzen [5] conducted a small pilot study in an advanced project-oriented software engineering course. Their results support the claim that TDD helps to improve both software quality and programmer confidence.

Other studies compared TDD to other processes using professional developers as subjects. Geras et al. [6] studied the differences between test-first and test-last concerning the amount of unplanned failures. Due to the small number of participating professional programmers the results of this study remain inconclusive. Canfora et al. [7] performed a controlled experiment with professional developers comparing TDD to a test-last approach. They report statistical evidence that TDD is more time-consuming but does not produce more assertions in the test code. George and Williams [8] compared TDD to a waterfall-like development process in a controlled experiment. The professional developers in their experiment worked in pairs. The programs of the pairs in the TDD group passed more test cases of a black-box test than the ones of the pairs in the control group. Yet, the TDD pairs also needed more time for development. A moderate but statistically significant correlation between the development time and the resulting code quality was found. Maximilien et al. [9] transitioned a software development team from an ad-hoc to a test-driven unit testing practice at IBM. While the transition to TDD had only a minimal impact on the productivity of the developers, the defect density of the project developed using TDD was reduced by 40 percent compared to a baseline project developed in a traditional fashion. Two case studies carried out at divisions inside Microsoft were reported by Bhat and Nagappan [10]. In each of the two divisions two teams worked on similar projects, one team using TDD and the other without using TDD. In both cases, the TDD group needed more time but the quality of the resulting software measured in terms of defects per LOC was at least two times better compared to similar non-TDD projects.

Research on the effectiveness of PP has produced significant results as summarized in a meta-study by Dybå et al. [11]. They analyzed the results of 15 studies comparing pair and solo programming and conclude that quality and duration favor PP while effort favors solo programming. Apart from the effectiveness of PP, other studies have taken an experimental approach to identify programmer characteristics critical to pair success. Domino et al. [12] examined the importance of the cognitive ability and conflict handling style. The performance of a pair was neither correlated with its cognitive ability nor its conflict handling style. Chao and Atli [13] first surveyed professional programmers to identify the personality traits perceived as important for PP. They then conducted an experiment with undergraduate students to identify the crucial personality traits for pair success that yielded no statistically significant results. Katira et al. [14] examined the compatibility of student pair programmers and found a positive correlation between the students' perception of their partner's skill level and the compatibility of the partners. Sfetsos et al. [15] compared the performance of student pairs with different Keirsey temperaments to student pairs with the same Keirsey temperament. The pairs with different temperaments performed better with respect to the total time needed for task completion and points earned for the tasks. The pairs with different temperaments also communicated more than the pairs with the same temperament.

3 Data Collection

We collected data from a total of 23 programmer pairs, nine in the novice '06 group, seven in the novice '08 group and seven in the expert group. Unfortunately, we had to remove the data points of five pairs from the data set: One data point in each group was dropped because the pairs did not manage to develop a solution which passed our automated acceptance test. Two more data points from the novice '06 group had to be excluded from the data set because their data was irreparably damaged due to technical problems with the Eclipse-plugins responsible for data collection. Hence, the following information refers to the participants whose data points are included in the data set.

3.1 Participants

The participants in the novice '06 and '08 group were students, who had participated in our extreme programming lab courses in 2006 and 2008, respectively. In the lab course, they learned the basic principles of extreme programming and applied them in a project week. Participation in the study was mandatory in order to obtain the course credits. The average member of the novice '06 group was in the 6.5th semester, had 4.8 years of programming experience, including 2.0 years experience with Java. Five members of the novice '06 group reported prior experience with PP, one had used JUnit before the lab course, and none had previously tried TDD.

In the mean, the members of the novice '08 were in the 7.7th semester, had 6.4 years of programming experience, including 4.0 years experience with Java. In the novice '08 group, three participants reported prior experience with PP. Three participants had used JUnit and three had tried TDD before the lab course. In both novice groups, industrial programming experience was sparse.

The members of the expert group were professional software developers from German IT companies, 11 from a company specialized in agile software development and consulting. One expert took part in his spare time and was remunerated by the experimenter, the others participated during normal working hours, and so all experts were compensated. All experts hold a diploma in Computer Science or in Business Informatics. On average, they had 7.8 years of programming experience in industrial projects including on average 5.5 years experience with PP, 3.0 years experience with TDD, 5.5 years experience with JUnit, and 7.2 years experience with Java.

3.2 Realization

For the assignment of the pairs in the novice groups the experimenter asked each novice for three favorite partners within their group and then assigned the pairs according to these preferences. Only one pair in each novice group could not be matched based on their preferences. In the expert group the pairs were formed based on their preferences and time schedule.

The task of the pairs was to complete the control program of an elevator system written in Java using TDD. The pairs received a program skeleton which contained the implementation of three out of four states. This skeleton comprises ten application and seven test classes with 388 and 602 non-commented lines of code, respectively. The set of unit tests provided with the program skeleton use a mock object [16] to decouple the control of the elevator logic from the logic that administrates the incoming jobs and requests. However, the mock object implementation in the skeleton does not provide enough functionality to develop the whole elevator control. Other functionality has to be added to the mock object to test all desired features of the elevator control. Thus, the number of lines of test code may be higher than the number of lines of application code. The mock object also contributes to the line count.

The pairs had to solve the task in a single programming session. All novice pairs and one expert pair worked in an office within the Computer Science department. For the other expert pairs an equivalent workplace was set up in a conference room situated in their company.

Because we were also interested in collecting data on the pairs' interaction all pairs worked on a workplace equipped with two cameras and a computer with screen capture software installed. There was an implicit time limit due to the cameras' recording capacity of seven hours. Additionally, the task description states that the task can be completed in approximately four to five hours. Each participant recorded interrupts such as going to the bathroom or lunch breaks. The time logs were compared to the video recordings to ensure consistency.

The pairs had to work on the problem until they were convinced they had developed an error free solution, which would pass an automated acceptance test, ideally at first attempt. If the acceptance test failed, the pair was asked to correct the errors and to retry as soon as they were sure that the errors were fixed. As mentioned before, one pair in each group did not pass the acceptance test after more than six hours of work and gave up.

4 Analysis and Results

The research hypothesis at the start of our data analysis was that the results of the expert group differ from the results of each novice group in terms of conformance to the TDD process and time needed for implementation. A comparison of the pre-test data of the two novice groups using a two-sided Wilcoxon test [17, pp. 106] revealed significant differences in two areas. The participants in the novice '08 group had studied longer ($p = 0.014$) and had more programming experience with Java ($p = 0.018$) than the participants in the novice '06 group. Therefore, we assumed that not only the expert group differs from both novice groups but that there could also be differences between the two novice groups.

To test this hypothesis for each metric, we applied a two step approach: As the sample size was small, we used a Kruskal-Wallis test [17, pp. 191] to test the null-hypothesis that all groups are equal against the alternative that at least two of the groups differ from each other. For the case that the Kruskal-Wallis test yielded a significant result, we used two-tailed Wilcoxon tests and Bonferroni-Holm correction [18] of the p-values to conduct the three possible pairwise comparisons of the groups. The significance level for all tests was set to 5 percent. Note that due to the small sample sizes the power of our study is restricted. We estimated the power of our tests on the basis of the power of the t-Test at a significance level of 5 percent and a large effect size of 0.8. Even if we do not use the rather pessimistic asymptotic relative efficiency (ARE) of 0.864 [17, pp. 139] but the more optimistic ARE value of 0.955 for normal distributions, the power of our study remains under 20 percent. That means that we have an about 80 percent chance of not detecting existing effects.

4.1 TDD Conformance

First of all, we examined the TDD conformance of the pairs. To obtain this measure, we used an advanced version of our TDD analysis framework first presented in [19]. Our framework consists of several Eclipse-plugins for data collection and a standalone analysis tool. The Eclipse-plugins register JUnit invocations, automated refactorings and frequently make copies of all changed Java files. The analysis tool uses the data collected by the Eclipse-plugins and extracts all changes made to application code methods and classifies them as test-driven, refactoring, or arbitrary. A change is classified as test-driven if it fulfills one of the following rules:

Rule 1: The change is contained in a method which was called by a failing test in the latest JUnit invocation before the point in time of the change.

Rule 2: The change is contained in a new method which is called by a failing test in the JUnit invocation immediately after the point in time of the change.

A change is classified as refactoring if it belongs to an automated[1] or a manual refactoring. The detection of test-driven changes and changes from automated

[1] As offered in Eclipse's Refactor menu.

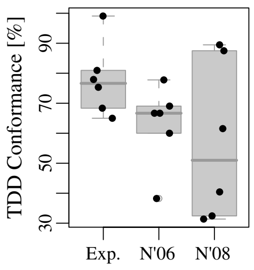


Fig. 1. Conformance to the TDD process

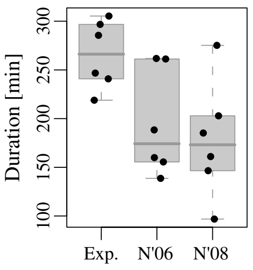


Fig. 2. Time needed for implementation

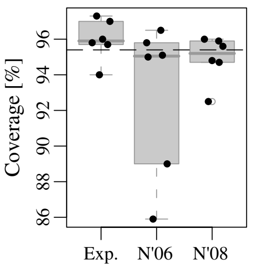


Fig. 3. Instruction coverage

refactoring works fully automatic. However, our analysis tool cannot detect manual refactorings automatically. Therefore, a human reviewer has to decide for all changes which could not be classified automatically whether they are refactorings or arbitrary changes, i. e. changes non-conformant to the TDD process. As the manual classification process is a potential source for errors, the analysis tool supports multiple reviewers and the reconciliation of reviews. For this study, we independently reviewed the changes and then reconciled the few conflicting decisions. Next, we computed the TDD conformance as follows:

$$\text{TDD Conformance} = \frac{\text{Test-Driven Changes} + \text{Refactorings}}{\text{All Changes}} \tag{1}$$

Fig. 1 shows the TDD conformance of the three groups as boxplots (grey) with the data points (black) as overlay. At first glance the TDD conformance in the expert group seems to be higher than in the two novice groups. However, the corresponding Kruskal-Wallis test returns a p-value of 0.203. Consequently, we cannot reject the null-hypothesis that all groups are equal.

4.2 Time

Next, we compared the time needed for implementation. It is defined as the time span from handing out the task description to the final acceptance test minus the time for the initial reading phase, breaks, and the acceptance tests. Fig. 2 depicts the time needed for implementation of all groups. Judging from the boxplots the expert pairs appear to be slower than the pairs in both novice groups. The results of the corresponding hypothesis tests as summarized in Table 1 confirm that the visible difference between the expert and the novice '08 group is significant. However, the differences between the expert and the novice '06 group, as well as between the two novice groups are not significant. The fact that the experts were slower than one of the novice groups was quite surprising, as we had assumed that the expert pairs would be faster than the pairs in the two novice groups.

4.3 Coverage

To assess if the expert pairs had invested the extra time in the quality of the unit tests, we used EclEmma to measure the instruction coverage, i. e. the percentage of instructions executed in a test run. Fig. 3 shows instruction coverage values of the final versions of the pairs' programs. The dashed line indicates the instruction coverage of the program skeleton originally handed out to all pairs. Though there seems to be a slight advantage for the expert group, the visible difference is too small to become significant (cf. Table 1).

4.4 Changes

In search of an explanation where the experts had spent their time we counted the number of lines that had changed from the initial to the final version of their program. Before counting the lines with the Unix diff-command, we standardized the formatting of all Java files with Jalopy and removed the import statements. Thus simple re-ordering of methods, changes made to white-space characters and import statements, as well as changes in comments had no effect on the number of changed lines. Fig. 4 depicts the results of this analysis. It appears as if the expert pairs had altered the code to a much larger extent than the pairs in both novice groups. However, according to the results from hypothesis testing as presented in Table 1 only the expert and novice '08 group differ significantly.

Remember, the difference in time consumption was also significant for the expert and the novice '08 group. So the fact that the expert pairs changed more lines of code than the pairs in the novice '08 group might be an explanation for the higher time consumption. Still, the question what led to more changes in the expert group needs to be answered.

Therefore, we take a more detailed look at the results we got from our analysis of the TDD processes: When we compare the percentage of changes that were classified as refactorings, we see that the expert pairs refactored significantly more than the pairs in both novice groups, whereas the difference between the novice groups is insignificant. A comparison of the number of automated refactorings used by each pair completes the picture (see Fig. 5). The expert pairs made significantly more use of Eclipse's refactoring capabilities than the pairs in the novice '06 and '08 group.

4.5 Threats to Validity

Apart from the different TDD and PP experience of the experts and novices other possible explanations for the observed differences in the data set might exist. First of all, the novices did not only have less experience with TDD and PP but also less general programming experience than the experts.

Secondly, the behavior of the participants might have differed from their actual development style since the participants knew that they were monitored by our Eclipse-plugins and cameras. In the post-test questionnaire[2], we asked the

[2] One expert pair had to leave before answering the post-test questionnaire.

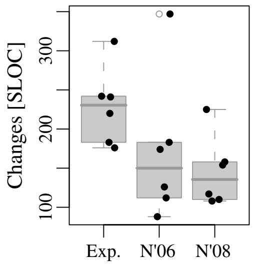


Fig. 4. Net number of changes

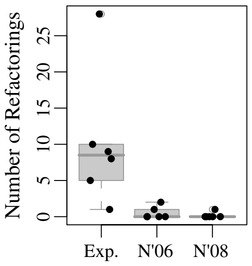


Fig. 5. Number of automated refactorings

Table 1. All p-values, underlined values are significant

Metric	*Kruskal-Wallis*	*Wilcoxon (Bonferroni-Holm corrected)*		
	All Groups	*Experts & Novices '06*	*Experts & Novices '08*	*Novices '06 & Novices '08*
TDD Conformance	0.203	-	-	-
Time	$\underline{0.039}$	0.130	$\underline{0.045}$	0.937
Instr. Coverage	0.232	-	-	-
Changed Lines	$\underline{0.049}$	0.217	$\underline{0.045}$	0.699
Auto. Refactorings	$\underline{0.003}$	$\underline{0.018}$	$\underline{0.014}$	0.527
Refactoring	$\underline{0.006}$	$\underline{0.019}$	$\underline{0.03}$	0.24

participants to rate the statement "I felt disturbed and observed by the cameras" on a Likert scale from 1 (totally disagree) to 5 (totally agree). About 65 percent of the participants rated this statement with totally or rather disagree.

Another threat to validity concerns the participants' motivation. The fact that experts were paid for their participation and novices were not might have led to a bias here. To check the participants' motivation, we asked them to rate the statement "I enjoyed programming in the experiment" on the same Likert scale mentioned before. Most participants (68 percent) totally or rather agreed with this statement.

Moreover, the participants' motivation might have been influenced by how well the partners got along with each other. We asked them to rate the statement "I would work with my partner again". The participants' agreement with this statement was even higher: About 82 percent rated it with totally or rather agree. We tested all of the groups' ratings on each statement for differences using a Kruskal-Wallis test. None of the results was significant.

Furthermore, the task was used in other studies before so some participants might have known the task. Consequently, we asked the participants if they already knew the task before they started. All participants replied with no.

Finally, some of the changes were classified manually. Manual classification might introduce bias into the data set. However, as mentioned in Section 4.1, both authors did the classification independently. Later on, the two reviews were reconciled. Only 3.8 percent of the changes were classified differently. Most of them were borderline cases where both classifications would have been applicable. Those differences could be resolved easily.

5 Conclusion

This article presented an analysis of the TDD conformance of 18 programmer pairs. First of all, we could neither provide evidence for a higher TDD conformance of the expert group compared to the novice groups nor for a better quality of the unit tests in terms of coverage. But since the sample sizes were small, this might be due to the low power of our study.

The most surprising result, nevertheless, is that the expert group was slower than one of the novice groups. However, the pairs in the same novice group had modified the given program skeleton less than the pairs in the expert group. Though not significant, the trend for the comparison of the expert and the other novice group pointed into the same direction. We could demonstrate that the plus in changed lines resulted from a higher fraction of refactorings in the TDD processes of the expert pairs. Additionally, they had applied more of the automated refactorings offered by Eclipse.

The results of this study have two implications: The first one is that studies with novices on the topics of TDD and PP are not easily generalizable to industrial settings. The second one concerns the low power of our study. Although it may be very tempting to use one-tailed hypothesis testing procedures to increase the power, they should be used with care. The direction of the difference is not necessarily the one predicted even if the underlying assumption seemed plausible when formulating the research hypothesis.

Experiments with sufficient power such as the study by Arisholm et al. [20] are rare in the research area of agile development. One way to overcome this problem is to collect data from several studies for meta-analysis. Yet, a better way, at least in our opinion, would be for different empirical research groups to join forces and set up larger experiments with potentially more valuable results.

References

1. Müller, M.M., Hagner, O.: Experiment about test-first programming. In: IEE Proceedings – Software, vol. 149, pp. 131–136 (October 2002)
2. Erdogmus, H., Morisio, M., Torchiano, M.: On the Effectiveness of the Test-First Approach to Programming. IEEE Transactions on Software Engineering 31(3), 226–237 (2005)
3. Pančur, M., Ciglarič, M., Trampuš, M., Vidmar, T.: Towards Empirical Evaluation of Test-Driven Development in a University Environment. In: EUROCON 2003. Computer as a Tool. The IEEE Region 8, vol. 2, pp. 83–86 (September 2003)
4. Edwards, S.H.: Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action. SIGCSE Bull. 36(1), 26–30 (2004)

5. Kaufmann, R., Janzen, D.: Implications of Test-Driven development: A Pilot Study. In: OOPSLA 2003: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 298–299. ACM, New York (2003)
6. Geras, A., Smith, M., Miller, J.: A Prototype Empirical Evaluation of Test Driven Development. In: Proceedings of the 10th International Symposium on Software Metrics, Washington, DC, USA, pp. 405–416. IEEE Computer Society, Los Alamitos (2004)
7. Canfora, G., Cimitile, A., Garcia, F., Piattini, M., Visaggio, C.A.: Evaluating Advantages of Test Driven Development: a Controlled Experiment with Professionals. In: ISESE 2006: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering, pp. 364–371. ACM Press, New York (2006)
8. George, B., Williams, L.: An Initial Investigation of Test Driven Development in Industry. In: SAC 2003: Proceedings of the, ACM symposium on Applied computing, pp. 1135–1139. ACM Press, New York (2003)
9. Maximilien, E.M., Williams, L.: Assessing Test-Driven Development at IBM. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, pp. 564–569. IEEE Computer Society, Los Alamitos (2003)
10. Bhat, T., Nagappan, N.: Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. In: ISESE 2006: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering, pp. 356–363. ACM Press, New York (2006)
11. Dybå, T., Arisholm, E., Sjøberg, D.I., Hannay, J.E., Shull, F.: Are Two Heads Better than One? On the Effectiveness of Pair Programming. IEEE Software 24(6), 12–15 (2007)
12. Domino, M.A., Collins, R.W., Hevner, A.R., Cohen, C.F.: Conflict in collaborative software development. In: SIGMIS CPR 2003: Proceedings of the 2003 SIGMIS conference on Computer personnel research, pp. 44–51. ACM, New York (2003)
13. Chao, J., Atli, G.: Critical personality traits in successful pair programming. In: Proceedings of Agile 2006 Conference, pp. 89–93 (2006)
14. Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Gehringer, E.: On understanding compatibility of student pair programmers. SIGCSE Bull. 36(1), 7–11 (2004)
15. Sfetsos, P., Stamelos, I., Angelis, L., Deligiannis, I.: Investigating the Impact of Personality Types on Communication and Collaboration-Viability in Pair Programming – An Empirical Study. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 43–52. Springer, Heidelberg (2006)
16. Mackinnon, T., Freeman, S., Craig, P.: Endo-testing: unit testing with mock objects. In: Extreme programming examined, pp. 287–301. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
17. Hollander, M., Wolfe, D.A.: Nonparametric Statistical Methods, 2nd edn. Wiley Interscience, Hoboken (1999)
18. Holland, B.S., DiPonzio Copenhaver, M.: Improved Bonferroni-Type Multiple Testing Procedure. Psychological Bulletin 104(1), 145–149 (1988)
19. Müller, M.M., Höfer, A.: The Effect of Experience on the Test-Driven Development Process. Empirical Software Engineering 12(6), 593–615 (2007)
20. Arisholm, E., Gallis, H., Dybå, T., Sjøberg, D.I.K.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering 33(2), 65–86 (2007)

An Exploratory Study of Developers' Toolbox in an Agile Team

Irina Diana Coman and Giancarlo Succi

Free University of Bozen-Bolzano
Via della Mostra 4, 39100 Bolzano, Italy
IrinaDiana.Coman@unibz.it, Giancarlo.Succi@unibz.it

Abstract. Although Agile teams supposedly value individuals and interactions over processes and tools, tools still represent an important support for developers' work. Existing studies investigate only partially tool usage in non-Agile teams. Moreover, it is not clear to which extent their findings are valid also for Agile teams. This study takes the first steps towards understanding tool usage in Agile teams by investigating the types and variety of tools used and the actual purpose for which they are employed. As expected, we found that communication accounts for an increased amount of time, but, surprisingly, a large share of it is represented by instant messaging or email rather than face-to-face communication. Other findings show that developers' toolbox contains only a very small number of tools and a relevant amount of time is spent on browsing the Internet and navigating through the file system.

Keywords: Agile Methods, Agile teams, work practices, tool usage.

1 Introduction

Agile teams value individuals and interactions over processes and tools[1]. However, Agile teams still recognize the need for good tools to support developers in their tasks. Examples of tools that directly support Agile practices are automated testing frameworks such as JUnit[2] or test coverage tools such as NCover[3].

Many tools are currently available to assist developers with their daily tasks. Many more tools appear constantly, claiming to better address developers' needs or to improve their effectiveness by integrating multiple functionalities or offering various services (such as component locating and testing [15]). Still, it is not clear to which extent this multitude of tools is actually used and to which extent it improves developers' effectiveness.

Existing studies offer strategies to assess and select a tool based on a combination of criteria such as tool usability, impact on developers' work or fitness for specific types of projects and development processes [1], [2], [3]. Such studies offer a way for selecting a new tool, but cannot guarantee its actual adoption

[1] Agile Manifesto http://agilemanifesto.org
[2] JUnit http://www.junit.org
[3] NCover http://www.ncover.com

and effectiveness. Thus, it is not clear whether developers would actually use the tool if they are given the choice and, if they do, whether they would actually use it exactly in the intended way, for the intended purpose and thus with the estimated impact on effectiveness.

Additional studies investigate non-Agile developers' cognitive processes [4], [5], work practices, and tool usage [6], [7], [8] with the main goal of proposing new tools that better support the observed activities. However, such studies do not directly evaluate the impact of existing tools and the way in which developers cope with the multitude of available tools. Moreover, it is not clear to which extent their findings apply also to Agile teams.

To evaluate the real impact of tools on developers' effectiveness, we need first to understand what tools developers actually use, to which extent and for which purpose. Existing studies do not directly address all these issues. This paper reports on the first steps towards addressing these issues by answering three main questions:

Q1. How many tools do developers actually use?
Q2. Which tools do developers use frequently?
Q3. Which main purposes serve the frequently used tools?

To answer the above questions, we perform a 53 days exploratory study in a small software team (3 developers) that is performing maintenance on several Web applications. As it is the first such study in an Agile team, the main goal is to shape some initial hypotheses regarding the toolbox of developers.

The main findings (to be tested also in other Agile teams) can be summarized as follows: Agile developers constantly try out new tools, but adopt only few of them; the toolbox of Agile developers contains very few tools (4) that are frequently used and a few more (7) for occasional use; Agile developers spend about 1/3 of their time on coding related activities, 1/3 on browsing the Internet, and 1/4 on communication.

This paper is structured as follows. Section 2 presents related work. Section 3 presents the research settings and section 4 presents the raw results. Section 5 discusses the results. Finally, section 7 draws the conclusions and identifies the limitations of this study and the directions for future work.

2 Related Work

Table 1 summarizes the findings of the existing studies on tool usage and work practices of developers. Although existing findings offer useful insights on the tool usage and work practices of non-Agile developers, it is not clear to which extent these findings apply also to Agile developers. While developers are reported to constantly try out new tools [7], it is not clear how many tools are actually adopted in the end.

A consistent finding over all the studies that investigated developers' activities seems to be that developers spend large shares (up to 50%) of their time on non-coding related activities [9], [7], [8]. However, there is not a clear consensus over the shares of various non-coding activities.

Table 1. An overview of empirical studies on developer activities and tools

Topic	**Purpose**	**Data collection**	**Findings on activities**	**Findings on tools**	**Ref**
Understand activities of developers	Improve tools	Survey, observation, interview	Most frequent: reading code, searching, editing, UNIX commands, compiling	Most used: compilers, search tools	[6]
Understand activities of developers and typical tools used	Improve tools	Surveys, interviews	Most frequent: communication, understanding and non-code, writing, editing. Developers constantly test new work practices	49% of developers use two or more tools when coding. Developers constantly test new tools.	[7]
Characterize how people interleave multiple tasks	Improve tools	Diary study	Most effort on: routine tasks, email, project tasks, task tracking	Not investigated	[8]
Study how developers spend their time	Minimize time loss in software development	Time diary	50% of the effort is spent on non-coding activities	Not investigated	[9]
Understand usage of software maintenance tools	Improve tools	Questionnaire, interview, direct observation, automated logging of tool usage	Developers explore software as much as edit it.	Most used: editors, search tools. Main positive: ease of use, useful features, rapid response time. Main negative: lack of integration, wrong mix of features.	[10]

All of the existing studies (with one exception, [10]) use various forms of manual data collection. However, manual data collection itself affects the work of developers, is error prone and does not allow for very fine grained analyses, at the level of short events (seconds) that might add up to important amounts.

By contrast to the existing studies, we investigate the tool usage in a new context, namely Agile teams. We focus on the actual tools that developers use on a daily basis and we try to understand which activities take large shares of developers' time. Unlike most of the existing studies, we use mainly automated data collection techniques that do not intrude upon the work of developers, are not biased, and offer very fine grained data accounting for each second of the time that developers spend interacting with their computers.

This study complements existing works in this field by investigating tool usage in the different context of Agile development. This study investigates also in more detail the toolbox of developers based on extensive, fine-grained data.

3 Research Settings

To answer our questions, we performed an exploratory study in a small Austrian software company that develops and maintains Web applications. Due to the sensitive nature of the data presented here, the company will remain anonymous and we'll refer to it as Company Small.

The team participating to our study consists in three regular developers. All three developers are Austrian, males, between 28 and 35 years old, with university degrees in computer related fields, and 5-10 years of programming experience. During our study, the developers performed maintenance tasks on three existing projects. They use Java, HTML, and Javascript as main programming languages and work with the Eclipse and Homesite5 IDEs. The team is an Agile team using a customized version of Extreme Programming [12], [13]. In particular, they use weekly iterations [13], [14] and user stories [14]. They are strongly encouraged to work not more than 8 hours per day. The characteristics of the development team involved in this study are summarized in Table 2.

Table 2. An overview of empirical studies on developer activities and tools

	Company Small
Team size	3 developers
Team type	Maintenance on web-applications (company's main product)
Nationality	Austrian
Background	University degree in computer related fields
Experience	5-10 years
Languages	Java, HTML, Javascript
IDEs	Eclipse, Homesite5
Paradigm	Customized XP (weekly iterations, user stories)

During all the duration of our study (53 working days), we use PROM [11] to collect data in a fully non-intrusive and automated way. The data consist in a stream of events that reflect the developer-computer interaction. Each event consists in a timestamp, the name of the currently used software application, and the name of the currently focused window (for all software applications) or the name of the current method, class, and file (for code). The granularity of the data is of one second. Developers could also manually insert data regarding time spent in non-computer activities such as meetings or phone calls.

The data collected from the three developers cover 52, 50, and 46 days respectively, as the developers were absent for various reasons during 1, 3, and 7 days respectively. On average, there are approximately 6 daily hours of computer interaction (6.62 ± 1.88), over all developers and all days.

4 Results

The goal of the study in company Small is to get an initial understanding of the toolbox of Agile developers, and to shape hypotheses regarding the potential answers to our initial three questions. Additionally, the study serves to identify other directions that were not obvious beforehand and that would benefit from deeper investigation.

To answer question Q1, we consider each distinct software application as a tool. Examples of tools are Internet Explorer browser or Eclipse development environment. We discard the tools that were used for less than 10 minutes (over all days and developers) as we consider that such a short usage time does not indicate actual usage or even testing of a tool. According to our data, the three developers used 26, 27, and 31 distinct tools respectively, during the 53 working days of the study. However, these tools were not all the same for all three developers. Considering the three developers together, they used 41 distinct tools.

As these numbers look rather big, we investigate more in detail how the total time recorded is divided among the usage of the various tools. There are only a few tools that have large shares of developer's time, while many others account for very little amount of time. Thus, to answer question Q2, we consider as frequently used tools only those that were used by each developer for at least 1% of his total time. According to our data, the three developers use frequently 12, 11, and 13 tools respectively. These frequently used tools account for 93.63% of the total time recorded for all three developers in company Small (Figure 1).

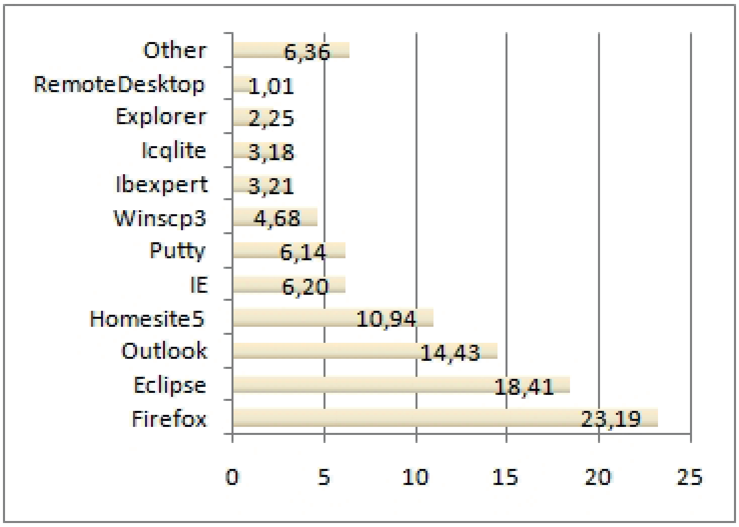


Fig. 1. Breakdown of computer interaction time on frequently used tools

Table 3. Categories and tools

Category	**Software tools**
Coding	IDEs; CVS/SVN clients; test frameworks; Internal software.
Communication	Email and Instant Messaging clients;Word(default editor for emails).
Internet	Internet browsers; Wiki tools.
Documents	All Microsoft Office except Word; all Open Office; text editors; Acrobat Reader; timesheet software.
Navigating	Windows Explorer; Microsoft Management Console; Task Manager; desktop search clients; console; archivers; file utilities.

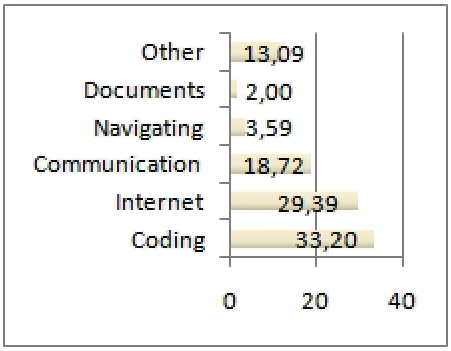


Fig. 2. Breakdown of computer interaction time on activities

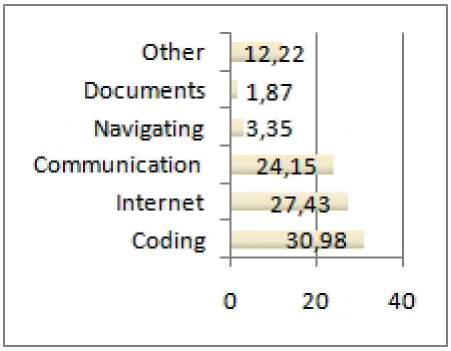


Fig. 3. Breakdown of 8 hours day on activities

As a first step towards answering question Q3, we group the tools into categories corresponding to higher level activities (Table 3). We obtain 5 main categories: Coding (includes IDEs and other software directly linked to programming activities, such as testing frameworks), Communication (email and IM clients as well as Microsoft Word as it was the default editor for email), Internet (Internet browsers and Wiki tools), Documents (Microsoft Office and Open Office), and Navigating (file system browsers and terminals). The tools that did not belong to any of these categories were all grouped under the name Other.

Computing the breakdown of developers' computer time among the 5 activities, reveals that Coding takes only 33%, followed closely by Internet (29%), Communication (19%), Navigating (4%), and Documents (2%) (Figure 2).

However, we compute these percentages considering only the computer active time. Considering all the 8 hours that make a developer's working day would decrease the percentage of time spent on Coding and would increase the percentage of time spent on Communication (as most of the activities outside the computer are planned or unplanned meetings and phone calls). A typical developer's day in company Small consists in 8 working hours out of which, on average, the developer spends 6.62 hours interacting with his computer. Adding thus the remaining time (on average 1.38 hours) as Communication and computing again the percentages considering 8 daily hours, yields that Coding accounts for only 31% of developers' time, followed by Internet (28%), Communication (24%), Navigating (3%), and Documents (2%) (Figure 3).

5 Discussion

Developers use a large number of tools overall (41), but less than a quarter of them on a regular basis (11). Most of the tools (30) are used only for very little time, accounting for less than 1% of the total time recorded. This would correspond more to developers exploring a tool rather than actually using it as part of their regular toolbox. However, in total, these tools account for 6% of developers' computer time which suggests that developers use about 1/2 hour daily to explore various tools. This would also be consistent with other existing studies that found that non-Agile developers are constantly testing new tools [7]. Considering also the small number of tools in the regular toolbox of developers (Figure 1), the conclusion is:

H1. Developers constantly try out new tools, but adopt only a few.

Out of the 11 tools in the developers' regular toolbox (accounting each of them for at least 1% of developers' time), there are 7 tools that are intensively used (each accounts for at least 5%), while the other 4 tools seem to be used only occasionally, accounting each of them for less than 5% (Figure 1). Thus, the second conclusion is the following:

H2. The regular toolbox of developers contains very few tools that are intensively used and a few more tools for occasional use.

The breakdown of developers' time on activities (Figure 2 and Figure 3) shows that developers spend only about 1/3 of their time on coding related activities while another 1/3 of the time they navigate the Internet and about 1/4 of the time they use communication tools. Additionally, only navigating through the file system or searching for a file takes about 4% of the time developers interact with their computer. Consequently, conclusions H3 and H4 are the following:

H3. Developers spend about 1/3 of their time on coding related activities, 1/3 on browsing the Internet and 1/4 on communication.
H4. Developers spend a relevant amount of their time ($\approx 5\%$) on navigating the file system or searching for a file.

The large share of Internet might be caused in this case by the fact that the team developed and maintained Web applications. Thus, part of the time spent on Internet might be due to testing the developed application and would rather belong to the Coding activity. This raises the need for deeper investigation of the Internet activity for identifying the different purposes of Internet usage. Thus, question Q3 would benefit from refinement with the following question:

Q3.1. For which main purposes do developers use Internet?

Although other studies already noticed that developers spend about half of their time in non-coding activities [9], and in particular in communication [7], these studies considered all the daily activity of developers rather than just their computer interaction. However, the study in company Small shows that communication maintains a large share of developers' time also when considering only the time spent in computer interaction. On one hand, this makes perhaps less surprising the large share of computer time (on average 6.62 daily hours out of 8) given that part of the communication seems to have migrated from other medias to the computer. However, the large share that communication has in computer interaction time is quite surprising considering also that the team was small and had the possibility of easy communication among the members given that they all worked in the same room. Moreover, as the team is an Agile team, the members supposedly valued, in all their interactions, face-to-face communication over other means. These observations lead to another question that details the main scope of question Q3:

Q3.2. For which main purposes do developers use communication tools?

To answer the additional questions Q3.1 and Q3.2, we have to distinguish between the various purposes for which the same application might be used. More precisely, the Internet usage could be further refined into searching for code examples or documentation, testing Web applications or editing the Wiki of the team. The communication tools could be used, for instance, for asking colleagues for information when getting stuck with some code (IM), for receiving company communications (email) or simply for taking a break. As these questions revealed themselves only at the end of this study, we do not have enough information to attempt to answer them for company Small. Instead, we consider that another study should be designed to collect all the data required to provide reliable answers to these questions and to test the conclusions of this study.

The implications of H1 and H2 would be that despite their big numbers and variety, most of the existing tools fail to convince the developers of their promised benefits. Developers seem instead to stick with the already established tools (Figure 1) and to prefer perhaps integrated functionality (such as plug-ins for the IDE) to new, stand-alone tools. Still, developers are in constant search of improved tools, as they diligently test many new tools (H1). As developers spend large amounts of time simply to find the files they need, there seems to be a dire need of improvement in the way in which files are organized on developers' computers. Current file systems seem to be inadequate for developers' needs and cost them precious time (H4).

The tools generally support developers in their activities, but it seems that some of them, such as Communication and Internet tools, might actually decrease developers' productivity. This is suggested by our findings that developers spend only 31% of their time coding (H3), while older studies reported 50% [9]. However, these implications need to be further tested in future studies.

Table 4. Main findings of this study

Literature	This study	Empirical conclusion
Developers constantly test new tools	Confirmed and added that few tools are adopted.	Most of the new stand-alone tools fail to convince developers.
50% of developers' time is spent for non-coding activities	66% of developers' time is spent for non-coding activities.	Part of the communication has moved to computer media; Internet and Communication might actually decrease productivity.
N/A	Few tools are actually used	Developers use established tools and prefer integrated tools.
N/A	$\approx$ 5% of developers' time spent navigating file systems or searching for files.	Current organization of file systems is inadequate for developers' needs.

6 Conclusions and Future Work

This paper offers an initial understanding of the actual toolbox of Agile developers, based on an empirical study in a software team of three developers. The study uses extensive data collected automatically and non-intrusively over 53 working days. The main findings are summarized in Table 4.

The collected data offer evidence of the following: developers constantly test new tools but adopt only few of them (H1); the regular toolbox of developers contains very few tools that are intensively used and a few more that are only occasionally used (H2); developers spend about 1/3 of their time on coding activities, 1/3 on browsing Internet and 1/4 on communication (H3); developers spend a large amount of time on navigating the file system or searching for a file (H4). Additionally, the study reveals a need to investigate deeper the usage of Internet and Communication tools and proposes to distinguish in future studies on various purposes for which developers use such tools (e.g. usage of Communication tools to ask colleagues for help or to receive company communications).

We consider as future work to perform other studies in Agile companies to further test H1 to H4 and to answer the additional questions Q3.1 and Q3.2.

When considering the findings of this study, several limitations have to be taken into account. The small size of the sample (one team of three developers) makes it hard to generalize the results. Thus, we consider the results of this study as initial concrete hypotheses to be assessed in future studies.

Additional internal limitations to this study concern the tool used for data collection. To ensure privacy rights, developers could suspend the data collection at any time, partially or completely, temporarily or definitively. Such option was transparent to us. However, the data cover on average 6 hours of daily interaction. As developers were rarely doing extra hours, we consider the 6 hours of data as a proof that developers did not suspend data collection for relevant amounts of time.

Another limitation is due to the data collection tool not being able to discern very short interruptions (seconds or a few minutes). Such interruptions can be easily generated as developers work in a shared environment. Thus, the active time spent in computer interaction might be slightly lower than the values observed in this study. Longer idle periods of time are not a threat to this study as the tool automatically detects and discards them.

References

1. Bruckhaus, T., Madhavji, N.H., Janssen, I., Henshaw, J.: The Impact of Tools on Software Productivity. IEEE Software, 29–38 (September 1996)
2. Poston, R., Sexton, M.: Evaluating and Selecting Testing Tools. IEEE Software, 33–42 (May 1992)
3. Atkins, D.L., Ball, T., Graves, T.L., Mockus, A.: Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor. IEEE Transactions on Software Engineering, 625–637 (July 2002)
4. Von Mayrhauser, A., Vans, A.: From Program Comprehension to Tool Requirements for an Industrial Environment. In: Workshop on Program Comprehension, pp. 78–86 (1993)
5. Von Mayrhauser, A., Vans, A.: From Code Understanding Needs to Reverse Engineering Tool Capabilities. In: Proc. Workshop on Computer-Aided Software Engineering, pp. 230–239 (1993)
6. Singer, J., Lethbridge, T., Vinson, N., Anquetil, N.: An Examination of Software Engineering Work Practices. In: Proc. CASCON, pp. 209–223 (1997)
7. LaToza, T.D., Venolia, G., DeLine, R.: Maintaining Mental Models: A Study of Developer Work Habits. In: Proc. of ICSE (2006)
8. Czerwinski, M., Horvitz, E., Wilhite, S.: A Diary Study of Task Switching and Interruptions. In: Proc. ACM Conf. on Human Factors in Computing Systems (2004)
9. Perry, D.E., Staudenmayer, N.A., Votta, L.G.: People, Organizations and Process Improvement. IEEE Software, 36–45 (1994)
10. Lethbridge, T.C., Singer, J.: Understanding Software Maintenance Tools: Some Empirical Research. In: Proc. Workshop on Empirical Studies in Software Maintenance (1997)
11. Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. In: Proc. of EUROMICRO, pp. 336–342 (2003)
12. Beck, K.: Extreme Programming Explained. Addison-Wesley, Reading (1999)
13. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2005)
14. Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley, Reading (2001)
15. Gross, H.-G., Melideo, M., Sillitti, A.: Self-certification and Trust in Component Procurement. Science of Computer Programming 56(1-2), 141–156 (2005)

Balancing Individual and Collaborative Work in Agile Teams

Hamish T. Barney[1], Nils B. Moe[2], Tore Dybå[2], Aybüke Aurum[1], and Martha Winata[1]

[1] Information Systems, Technology and Management, UNSW, Sydney NSW 2052, Australia
hamish@hbarney.com, aybuke@unsw.edu.au, m.winata@student.unsw.edu.au
[2] SINTEF ICT, NO-7465 Trondheim, Norway
{nilsm,tored}@sintef.no

Abstract. In the agile approach, the self-organizing team itself decides how work is coordinated. For individuals in a team to be motivated and satisfied with their job they need to have control over their work and over the scheduling and implementation of their own tasks. However, individual and team level autonomy may conflict, and reduce the effectiveness of the team. Therefore, there is a need to investigate how to achieve empowerment at the individual and team levels simultaneously. An Australian software developer, Atlassian, has developed an interesting way of solving these problems with FedEx Day. Once every three months, developers get a day to work on whatever they like! Like an express courier, a developer must deliver something in a day. Developers then present their work to the rest of the company. Some of this work then ends up getting incorporated into the products.

Keywords: teamwork, individual work, autonomy, self-managing, self-organizing, agile software development, scrum, XP, FedEx day, case study.

1 Introduction

Software development requires the intellectual capital of skilled professionals – both as individual contributors and as collaborators in a team, where the knowledge of several individuals is integrated. Such teamwork has become critical for software companies, with team autonomy as a crucial factor for work group effectiveness [1]. At the same time, individuals need to be motivated and satisfied with their jobs by having control over their own work and over the scheduling and implementation of their own tasks [1-3]. However, as individual work becomes more autonomous, there will be less interaction between group members, which may be a threat to teamwork effectiveness [1].

Therefore, there is a need to balance both individual and team autonomy in a software team. Hoegl and Parboteeah [4], for example, found that software developers doing innovative work need to participate in both collaborative work periods and individual work periods. Also, research in psychology and organization has identified individual autonomy as an important intrinsic motivator [2, 3, 5]. Unfortunately, using individual autonomy as such a motivator is fraught with a range of difficulties. How can a company ensure that employees given the freedom to work independently will not squander the opportunity or worse yet use the company's assets for personal gain at the expense of the company?

Team autonomy is at the heart of agile software development in which the team itself decides how work is coordinated [6]. In Scrum, for example, the team is given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions.

It is worth noting that while agile practices, like Scrum and XP, often give the team as a whole a great deal of autonomy, it does not follow that the individual team members are given high levels of individual autonomy. Barker [7], for example, pointed out that self-managing groups may end up controlling group members more rigidly than they do under traditional management styles, while Markham and Markham [8] suggested that it may be difficult to incorporate both individual autonomy and group autonomy in the same work group.

Thus, balancing team level autonomy and individual level autonomy in software teams is challenging; especially when development is done in market-driven agile projects with fixed scope and deadlines. Motivated by this, our research question has been: *How can individual and team autonomy be balanced in a software team?*

We have studied a company, Atlassian, that has found one way of balancing these levels of autonomy by implementing an organizational practice that fosters both collaborative and individual work and that also motivates their employees to work in the interest of the firm without diminishing intrinsic motivation "FedEx Day".

The rest of this paper is organized as follows: Section 2 gives a overview of research on collaborative and individual work. Section 3 defines and describes the case-study. Section 4 presents our findings, and finally, our discussion and conclusions are presented in section 5 and 6 respectively.

2 Collaborative and Individual Work

Although agile development is a new approach to organization, the notion of self-management is not new; there has been research in this area since Trist's examination of self-regulated coal miners in the 1950s [9]. We use the label "self-organizing" teams as a synonym for "autonomous teams" and for "empowered teams". Guzzo and Dickson [10] described such a team as a group of employees who typically perform highly related or interdependent jobs, who are identified and identifiable as a social unit in an organization, and who are given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions with economic consequences.

Autonomous teams stimulate participation and involvement, leading to team members developing an emotional attachment to the organization, greater commitment and motivation to perform and desire for responsibility [11]. Self-organization requires a belief in local rationality of individuals and units (e.g. those closest to the customer know the customer best). It is consistent with the often espoused idea of delegating decision making to the lowest possible level and it implies maximizing capabilities of scope at every level of the organization [12]. Self-management can also directly influence team effectiveness since it brings decision-making authority to the level of operational problems and uncertainties and, thus, increases the speed and accuracy of problem solving [13].

Even though there are several studies on the benefits of self-organizing teams, there is substantial variance in research findings regarding the consequences of such teams on such measures as productivity, turnover, and attitudes [10, 13]. Tata and Prasada [13] found that employees really need to affect managerial decisions in order to achieve the putative benefits of a self-managed team. It is important that the team does not experience autonomy that is only symbolic.

The fact that autonomy can simultaneously reside at both the group and the individual level in a work group is often neglected in studies of self-management [1]. A group may have considerable discretion in deciding what group tasks to perform and how to carry them out, but individual members within the group may have very little discretion or control over their jobs.

Encouraging individual autonomy has been shown to have many positive effects in both laboratory and field research [2]. Offering workers greater individual autonomy increases: the degree to which extrinsic motivation offered employees is internalised [14], job satisfaction and positive work attitudes [15-17]. Greater internalisation of extrinsic motivation has been shown to lead to effective performance on tasks requiring creativity, cognitive flexibility and conceptual understanding [18-20]. These are the very characteristics of software development work. Practices that increase individual autonomy are, therefore, extremely valuable, especially if they can be leveraged to provide real value to the employer.

Kirkman and Rosen [21] emphasized the importance of taking both individual and group effects into account, stating that ”what is needed most in the team effectiveness literature is research that examines empowerment at the individual and team levels simultaneously.” Additionally, a practical implication of Langfred’s [1] findings is that, if an organization believes in letting teams be more self-managing, great care must be taken in the implementation. Autonomy at the individual level may conflict with autonomy at the group level, producing countervailing influence on the cohesiveness and, indirectly, effectiveness of the team. An organization could thus experience little or no results from empowering employees. As far as the authors are aware, there is no empirical work from the software industry on how the balance between individual and team autonomy can be achieved.

3 Research Method

Given the focus discussed above, we choose a case study that would allow us to investigate our research question.

3.1 Study Context

The case study was undertaken at Atlassian, a medium-sized company selling mass-market software. The software is also used internally in Atlassian, which means that the developers are also users of the software. This is seen as a positive thing within the company and something that may be important in facilitating a practice like FedEx Day. For instance, the program manager commented that “the fact that we use our own products to help communicate and to help foster that culture is a really cool reinforcing kind of loop and it means that everyone owns their own products and we all use [the Atlassian products] internally, so it means you can be really proud of a product… In some of the companies I previously worked at I’ve had no idea how to use our products or what the typical user’s problems might be”.

The company is based in Australia but has offices in several countries in Europe and the United States. Software development is undertaken at all of these locations. The company has, since its inception, used a combination of XP and Scrum, and has undergone rapid growth; approximately doubling its size each year of its seven year existence.

Atlassian is an open company with important company details being available to all employees. Internal and external wikis and blogs are used heavily and these often host lively discussions about the company. As one of the managers commented, “everything gets documented on the [intranet], everyone has buy in and everyone has a say in everything so that’s why it’s such a cool place to work”.

The founders of the company were responsible for the initial development of some of Atlassian’s products. They still play a role in software development and are well-known within the company for quickly developing prototypes of new features. When interviewed, the head of engineering commented on this practice: “Yes, especially [one of the founders]. But he's prototyping, he'll be: I can't tell you what I want, so let me code it real quick, then I’ll show you, that's [one of the founders]. Yeah it's scary.”

The attitude displayed by the founders towards prototyping has translated itself throughout the company to a preference for action rather than just words or ideas. One of the core company values is for individuals within the company to be proactive, not just to have ideas but also to do something about them. A tech lead discussed this aspect of Atlassian’s culture: “it's just ideas and I have ideas and I want other people to see my ideas, our company isn't as big on it. It's basically do it. Don't just tell me about some great dashboard, show me a prototype, do something, make it happen in the product or something.”

3.2 Data Sources and Analysis

We relied mainly on qualitative semi-structured interviews as these provide a rich picture of the internal workings of the company in general and the specifics of the FedEx Day, the development practice under investigation. 17 employees were interviewed: 15 team members, tech leads, and team leads from three product teams and two executives. Each of the semi-structured interviews took approximately one hour. All interviews were recorded and transcribed.

To be able to address a broader range of historical and behavioral issues [22], we used multiple sources of evidence. In addition to interviews, we collected data from Atlassian’s internal and external websites, which host information and discussions about Atlassian’s development practices and the company’s structure and culture. These sites are updated frequently and all employees are encouraged to participate in these forums. Data were then categorized and coded. Observations were also made in-situ by attending meetings and observing the operation of FedEx Day.

By combining the data from interviews with the information from the websites, we were able to develop a converging line of inquiry [22] by forming a rich and accurate picture of the company in general and the practice of FedEx Day in particular. Our main analysis technique was to combine a pattern-matching logic with explanation building [22]. That is, we compared the empirically based patterns with the ones predicted from the theoretical propositions of team level and individual level autonomy, while at the same time building an explanation of the case. This strategy also helped us to strengthen the internal validity of the case study.

4 FedEx Day and Agile Development at Atlassian

4.1 Task Allocation at Atlassian

The way tasks are allocated normally at Atlassian is crucial to understanding FedEx Day and its effects. Each team has a product backlog that is updated regularly based on *releases*. A release includes major feature improvements that can be marketed and distributed to all customers. Features in the release backlog get divided into sprints and each sprint produces a point release – a smaller product release. At the beginning of each sprint, the list of features is presented to the developers.

Identifying and allocating tasks/features affects the level of individual autonomy since this defines what the developer will do. This is done differently in the various teams at Atlassian. We found that in some teams there was little scope for individuals to choose the tasks they wanted, because there was little redundancy in the teams. One developer said: “A lot of the time it is based on whoever can do the tasks because we've got all very different skill sets”. One team leader from another project said: “People do volunteer, but there's never a surprise on what they volunteered for.” Another said: “we don’t share what everyone’s working on and how it works. That’s why if someone gets sick, it can take longer time to pick up the stuff after them”.

Another team leader, responsible for 15 developers in 4 sub-teams, said: “We found with XP, the whole approach of not doing a lot of planning up front gives us a lot of trouble… a lot of people get frustrated because the release takes longer than the original plan”. This team leader pre-planned the sprint with management and with some input from developers. Then, during the Scrum planning meeting, the team leader assigns tasks to developers.

The introduction of story cards into one of the development teams was seen as reducing individual autonomy. Story cards are a standard XP practice. Requirements are broken up into short user stories, which are written on small cards. A developer is assigned (or chooses) one or more story cards to work on. Estimated and actual time spent is also tracked on the story card. In this team story cards were assigned to developers in the fortnightly planning meetings or by the team leader. In many ways the introduction of story cards increased group autonomy with story cards being selected and prioritised by the whole team during their planning meetings. However, many developers commented on the feeling that, with their time being tracked, they no longer had the freedom to experiment and play with new features.

The developers are encouraged to suggest features that should be in the product backlog. However, resource limitations led upper management to reduce the number of features the teams could develop for a release, since resources are allocated according to the revenue generated. In terms of prioritising the features, a developer explained how he influenced the direction of the product: "[The CEOs] do kind of get a higher priority when we're making our decisions. But in the end it's really the developers and our product manager that decides what goes in and what goes out and have full control over what we do". Only a few developers reported having this kind of influence on the product, mostly those in the smaller teams.

4.2 FedEx Day

FedEx Day is one of the practices that Atlassian has adopted that allows the developers to exercise their individual autonomy in developing software in a way that also has direct benefits for Atlassian. Once, every three or four months, most developers voluntarily take a day to work on whatever they like. Just like an express courier, a developer has to deliver something in a day. After they are finished they present and demonstrate their work to the rest of the people in the company. Some of this work then ends up getting incorporated into the products. FedEx Day is not just a one (or one-and-a-half) day affair. There is actually a couple of weeks of preparation that go on leading up to a FedEx Day and then, after FedEx Day, some of the finished projects are made production-ready.

A FedEx Day coordinator does a lot of the organization that goes into making a FedEx Day a success both before and after the event. The FedEx Day coordinator lets everyone know that a FedEx Day is coming up. He or she organizes a series of voluntary lunchtime meetings where developers bring their lunch, to discuss FedEx Day project ideas. Most people already have some kind of idea about what they want to do, for instance: something about the product that's been bothering them for months, some new technology that they want to try out or some new feature that they think would be brilliant but cannot get other people to agree. Some developers go to the meetings not quite sure what they want to do, looking to be inspired by other people's ideas. Developers are not restricted to working on the product they normally work with, "it can be anything you want." (Developer).

As FedEx Day approaches, developers write up their FedEx Day ideas as "FedEx Orders", which are short descriptions of what they aim to achieve. Other developers comment on these "FedEx Orders", offering hints, suggestions and ideas.

Watching the developers work during FedEx Day, the sense of excitement and focus is palpable. People work more intently for the time during FedEx Day than at any other time. Developers described the adrenaline and sense of exhilaration and gut wrenching pressure they feel when they are competing in a FedEx Day. The whole time the ticking clock hangs over them like the sword of Damocles. Developers feel a genuine sense of pressure, they want to show off their abilities and prove that their ideas *will* work.

Instead of their usual development process that includes producing automated tests and documentation, "[y]ou just blast out the feature, hack it out however you want" (Developer). This offers what developers perceive as a pleasant change of pace from the usual, and more conservative, day-to-day development process. These things are added later if the feature is adopted in one of the products.

After a full day of intense activity, the developers present their projects to one another. Experienced FedEx Day developers also know that an impressive presentation can make or break a FedEx Day project. As a result they always make sure that they set aside time to create a convincing presentation, even if it means that the last one or two features aren't quite finished. One of the veteran team leaders commented that: "the easiest way to impress people [in FedEx Day] is to develop something flashy… if the demo looks good… it works out."

After watching everyone's presentation, people vote on which projects they thought were the best. The winner gets a prize: an attractive trophy (with FedEx vans on it) and, perhaps most importantly, bragging rights for the next couple of months.

In the weeks following FedEx Days, developers write blog posts about their FedEx projects (sometimes with some gentle prodding by the FedEx Day coordinator). The FedEx Day coordinator then organizes the publication of some of these FedEx Day Project write-ups on Atlassian's external blogs. Some of the projects then make it in to the product backlogs. Sometimes the FedEx Day project will be included as optional extras that customers can download.

4.3 Benefits of FedEx Day

According to one team leader, developers "get to work on things that they really want to do". The developers themselves share this perception. For instance one developer commented that he "love[s] doing [FedEx Day] because it is something that I thought of myself and it's something that hasn't really been done before." It provides developers with a "creative outlet" that some developers feel they need.

As a result of FedEx Day, the developers "make almost individual contribution to the product, makes them identify with the product a lot more". Another team leader's statements provided another example of the perceived benefits: "I think the best thing about FedEx is developer morale, even if we don't get a lot of usable products out, and we do, but even if we didn't, the benefit to morale is incalculable, because people get to go and do fun stuff that stops burnout, it lets people play with new ideas, it lets everybody know that their ideas and their thought processes are valued, and so I think that's the best, people get the chance to drop everything else, do something fun, and Atlassian gets value out of it, because the stuff that comes out of there are connected, and can shift [the direction of the product]."

The effect on productivity that providing people with greater individual autonomy predicted by some of the psychology literature [18-20] is also confirmed by the experience of FedEx Day at Atlassian. One team leader estimated that "people get about three days of work done in that one day, it's amazing". The intensity with which people work and their productivity was confirmed by our own observations.

Most of the interviewees cited the degree of participation in the decision-making process, with respect to decisions about the products that they are working on and the company as a whole, as one of the best things about working at Atlassian. Interestingly, this was despite the observations that individual developers did not seem to exercise a much direct influence over the selection and prioritization of requirements, especially in the larger teams. One developer said that the best thing about working at Atlassian was he is "part of that decision making process that decides what goes into the product. So I feel that I also have control over what my product does, and what my work is going to be." Similarly, another developer commented that "I've never really felt that there is a huge impedance between things that I want to do and where the company is going." FedEx Day appears to play a role in fostering this feeling among the Atlassian employees.

4.4 FedEx Day Challenges

While all interviewees were in agreement that FedEx Day is a net positive there are some negative aspects to FedEx Day. For instance, the competitive aspect of FedEx Day means that crowd-pleasing presentations and visually impressive features will often win over features that, while more technically interesting, are less eye-catching. Two team leaders commented on this phenomenon: "the problem with FedEx I think it's the fact that it's a competition right? …I guess the easiest way to impress people [and win FedEx Day] is to develop something flashy", "usually just some[thing] really flashy [wins]". It should, however, be noted that this is not always the case. Many FedEx Day projects are useful and are integrated into the products.

The expense and difficulty of arranging for all developers to take a day and a half off normal work to participate in FedEx Day is also one of the challenges associated with FedEx Day. Despite an upcoming scheduled release of a product, the team lead felt obliged to encourage the developers in his team to participate in FedEx Day despite the difficulties that could have ensued if the release went badly.

5 Discussion

This study highlights the difference between team and individual level autonomy, which is especially important in the context of agile software development. In this case study it has been shown that individual level autonomy can sometimes be suppressed even in self-managing teams. As such, it is important that compensating practices be adopted that allow companies to strike a balance between individual and team autonomy. One particular practice that acts to promote individual autonomy, FedEx Day, was introduced at Atlassian.

Past research has shown that people working in self-organized teams are more satisfied [11]. However, working in a self-organized team may not lead to employees fulfilling their needs for individual autonomy. For instance, one important aspect of working in an agile team is getting all team members to commit to the requirements that are to be implemented and their priority. Achieving this agreement may lead to a suppression of individual autonomy as individuals' ideas and interests are suppressed in process of gaining shared commitment.

The fact that tasks were often assigned based on skills rather than preferences was the main reason for the reduction in individual autonomy in Atlassian. Morgan [23] refers to this as lack of redundancy. FedEx Day is one way that Atlassian addresses this reduction in individual autonomy, allowing developers to experiment with technologies and ideas that they may be interested in but rarely get to try because of their existing skill-set.

Another finding was that the introduction of story cards in the development process was associated with greater team-level autonomy but reduced individual autonomy. By adopting practices like FedEx Day, organizations may be able to counteract the reduced individual-level autonomy that many of the popular XP and Scrum practices may lead to. FedEx Day allowed developers to explore and experiment with new technologies and their own ideas that the adoption of story cards had suppressed.

Despite the fact that most developers are not directly involved in the decision-making process about the selection and prioritization of features they still feel very involved in the decisions that are made in the company. This perception can, to a great extent, be attributed to the practices that foster individual autonomy, like FedEx Day. FedEx day is an interesting practice that facilitates greater individual autonomy. Increased levels of individual autonomy have been shown to have a range of beneficial effects especially with tasks that, like software development, require creativity, cognitive flexibility and conceptual understanding [18-20].

6 Conclusion

In this article we presented FedEx Day, which is one way Atlassian has addressed the challenge of balancing individual autonomy with high levels of team autonomy. By setting aside a day and a half for developers to work on a project of their own choosing, Atlassian has been able to increase the level of individual autonomy offered to their staff. A range of benefits have accrued to Atlassian as a result of adopting this practice including: a positive effect on staff morale, advantages in hiring and retaining staff, greater identification by the developers with the company and its products and increased productivity. FedEx Day has several additional benefits for the company and the individuals that participate, including encouraging innovation and entrepreneurship. There are also potential negative impacts of FedEx Day like competitiveness and poor self-esteem. These did not become apparent over the course of this study but may emerge in a longer-term study of this practice. We plan to examine these aspects of FedEx Day and related practices in our future research.

It would also be of great interest to assess the impact of FedEx Day in other organizations. The effects of adopting this practice in other organizations could be undertaken either by engaging in action research or by identifying organizations that engage in similar practices.

References

1. Langfred, C.W.: The paradox of self-management: Individual and group autonomy in work groups. Journal of Organizational Behavior 21, 563–585 (2000)
2. Gagne, M., Deci, E.: Self-determination theory and work motivation. Journal of Organizational Behavior 26, 331–362 (2005)
3. Osterloh, M., Frey, B.: Motivation, Knowledge Transfer, and Organizational Forms. Organization Science 11, 538–550 (2000)
4. Hoegl, M., Parboteeah, K.P.: Creativity in innovative projects: How teamwork matters. Journal of Engineering and Technology Management 24, 148–166 (2007)

5. Foss, K., Foss, N., Klein, P.: Original and Derived Judgment: An Entrepreneurial Theory of Economic Organization. Organization Studies 28, 1–20 (2007)
6. Boehm, B.W., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley, Reading (2003)
7. Barker, J.R.: Tightening the Iron Cage - Concertive Control in Self-Managing Teams. Administrative Science Quarterly 38, 408–437 (1993)
8. Markham, S.E., Markham, I.S.: Self-management and self-leadership reexamined: A levels-of-analysis perspective. The Leadership Quarterly 6, 343–359 (1995)
9. Trist, E.: The evolution of socio-technical systems: a conceptual framework and an action research program. Ontario Quality of Working Life Centre, Ontario (1981)
10. Guzzo, R.A., Dickson, M.W.: Teams in organizations: Recent research on performance and effectiveness. Annual Review of Psychology 47, 307–338 (1996)
11. Parker, S., Wall, T., Jackson, P.: That's Not My Job: Developing Flexible Employee Work Orientations. Academy of Management Journal 40, 899–929 (1997)
12. Volberda, H.W., Lewin, A.L.: Co-evolutionary Dynamics Within and Between Firms: From Evolution to Co-evolution. The Journal of management studies 40, 2111 (2003)
13. Tata, J., Prasad, S.: Team Self-management, Organizational Structure, and Judgments of Team Effectiveness. Journal of Managerial Issues 16, 248–265 (2004)
14. Deci, E., Eghrari, H., Patrick, B., Leone, D.: Facilitating Internalization: The Self-Determination Theory Perspective. Journal of Personality 62, 119–142 (1994)
15. Baard, P., Deci, E., Ryan, R.: Intrinsic Need Satisfaction: A Motivational Basis of Performance and Weil-Being in Two Work Settings 1. Journal of Applied Social Psychology 34, 2045–2068 (2004)
16. Deci, E., Ryan, R., Gagne, M., Leone, D., Usunov, J., Kornazheva, B.: Need Satisfaction, Motivation, and Well-Being in the Work Organizations of a Former Eastern Bloc Country: A Cross-Cultural Study of Self-Determination. Personality and Social Psychology Bulletin 27, 930 (2001)
17. Gagne, M., Koestner, R., Zuckerman, M.: Facilitating Acceptance of Organizational Change: The Importance of Self-Determination 1. Journal of Applied Social Psychology 30, 1843–1852 (2000)
18. Amabile, T.: The social psychology of creativity: A componential conceptualization. Journal of Personality and Social Psychology 45, 357–376 (1983)
19. Grolnick, W., Ryan, R.: Autonomy in children's learning: An experimental and individual difference investigation. Journal of Personality and Social Psychology 52, 890–898 (1987)
20. Benware, C., Deci, E.: Quality of Learning With an Active Versus Passive Motivational Set. American Educational Research Journal 21, 755 (1984)
21. Kirkman, B.L., Rosen, B.: Beyond self-management: Antecedents and consequences of team empowerment. Academy of Management Journal 42, 58–74 (1999)
22. Yin, R.K.: Case study research: design and methods. Sage, Thousand Oaks (2009)
23. Morgan, G.: Images of Organizations. SAGE publications, Thousand Oaks (2006)

Organizational Enablers for Agile Adoption: Learning from GameDevCo

Jayakanth Srinivasan and Kristina Lundqvist

Mälardalen University, Box 883, 721 23 Västerås, Sweden
{jayakanth.srinivasan,kristina.lundqvist}@mdh.se

Abstract. Adopting agile methods requires an understanding of both the mechanics and the dynamics of value creation in software organizations. From a mechanics perspective, successful agile adoption is about ensuring that project stakeholders are aligned toward a common project objective, employees have the ability to make decisions at the right level of abstraction, that there is effective project management, and an environment exists that supports individual and group learning. The dynamics of value creation require an understanding of organizational-level stakeholders and their value propositions, the development of an organizational learning system, and last but not least, an effective governance strategy. This paper uses the lessons learned a case study of GameDevCo to illustrate these organizational enablers for agile adoption.

Keywords: Agile Adoption, Case Study, Engaged Scholarship, Enablers.

1 Introduction

Software organizations are in a knowledge-intensive industry, within which they have to deal with the rapid changes in technology, unpredictable variation in customer values, and increasing competition for their human capital. Given that software development is a non-routine complex undertaking requiring high levels of competence and a flexible organizing structure, the fundamental issue for software organizations is how to achieve a balance between control and goal orientation on one hand, and change and flexibility on the other[1]. Teece, Pisano and Shuen [2] define dynamic capabilities as , 'the firm's ability to integrate, build and reconfigure internal and external competencies to address rapidly changing environments'. Zollo and Winter [3], further extend this definition of dynamic capabilities as, 'a learned and stable pattern of collective activity through which the organization systematically generates, and modifies its operating routines in pursuit of effectiveness'. The notions of learning, collective activity and effectiveness are important in the software context: the activity in software organizations is organized through the use of project teams, hence through collective activity; an effective software organization is one that can deliver its products and services to its customers on schedule, within budget and at the highest quality even when operating in a changing environment.

In looking at the software services industry, Ethiraj et al. project-specific capabilities and client-specific capabilities both contributed to project success [4]. They note that client-specific capabilities reflect the tacit knowledge of a client's business domain and operating routines, and are obtained through repeated interactions with a client across multiple projects over time; while project-management capabilities are acquired through a deliberate and persistent investment in infrastructure and training. Athreye's study of the Indian software industry [5], highlighted the importance of aligning capability development to the business model. When looking at organizations that have been successful at generating capabilities through improvement efforts, two stand out – Toyota in the automotive industry, and Rockwell Collins in the -aerospace industry. While the Toyota Production System [6, 7] has been widely touted as the source of Toyota's competitive advantage [7], a deeper look reveals that their philosophy of 'respect for people' [8, 9] plays an equal if not greater role in their success. Their True North metrics [10] of human development, quality, cycle time, and cost/productivity; in that order, highlight the importance of people in generating organizational capabilities. The use of the Lean Electronics program at Rockwell Collins to drive continuous improvement is very well documented [11], but the true source of their competitive advantage lies in their ability to create an effective value proposition for people. The synergy of having an improvement program, coupled with the ability to motivate process users to drive the effort, enables these organizations to develop enterprise-level agility. This paper addresses the question of:

What are the organizational enablers that support the adoption of agile methods in software enterprises?

The remainder of the paper is organized as follows: first, we present the context within which the GameDevCo case study was carried out, followed by the analyses across the organizational enablers of stakeholder alignment, employee employment, group & organizational learning, and governance mechanisms. It is important to note that these categories emerged from the analysis of the interview and observation data. The paper concludes with a discussion of the findings, and implications for future work.

2 Case Context

The primary focus within this paper is on GameDevCo, which was born as a startup project that was designed and built on a university campus by developers who had a passion for the game of poker, and deep technological expertise in all three critical areas: server-side software, game engine, and client-side software. The game was initially designed to develop the game playing skills using 'play' money, but the success of the game overall, led to the creation of a product that would allow people to buy-into and play the actual games. The market success led to the team starting a company. GameDevCo's senior leadership team noticed that software was not being developed on-time, and on-cost. To add insult to injury, the constant bug-fixes and patches were being implemented in the 'live environment'. As a result, they imposed a very structured development process on the team based on the heavily plan-based development approach adopted at a major telecommunications company. The structured development process really did not improve organization performance, and given that the development team was improving the product incrementally, GameDevCo's senior leaders felt that the developers should be provided with the autonomy needed to make the decisions needed, while ensuring that some formal product data management was being carried out. Five years after their inception in 2000, they decided to adopt Scrum as the development methodology of choice.

Table 1. Interviews at GameDevCo

Senior Leader	Business Unit Head (3), Senior Architect (1), Testing Head (1)
Middle Manager	Scrum Masters (5), Senior Designers (2), Product Owner (1)
Workforce	Developers (4), QA (2), Graphic Designers (3)

As part of the research for this paper, in late 2007, we first interviewed 22 people (including three teams) who spanned the organization hierarchy from senior leaders to developers and testers, as shown in Table 1. We followed up with a second round of interviews in mid 2008, followed with a meeting of the senior leadership team in Jan 2009, to increase the trustworthiness of the analysis.

3 Stakeholder Alignment

From the perspective of achieving stakeholder alignment in software projects, the best known model is the win-win spiral model [14]. As Boehm points out [15], the emphasis of stakeholder commitment to shared systems objectives provides the organization with a collaborative framework for helping people and organizations cope with change. The importance of software organizations managing the needs of their key stakeholders has been emphasized at the project level from the perspectives of architecture [16], requirements management[17], globally distributed development [18], just to name a few. The win-win spiral model however, does not explicitly address the organization level stakeholders. An extension to the model used in this paper, accounts for the fact that software organizations have to necessarily take a bi-level perspective, addressing the needs of senior leadership, and customers (both internal and external) at the organizational level, and meeting the needs of customers, developers, architects, users and maintainers at the project level. In effect, the model views a software enterprise as a portfolio of projects, that each has a set of stakeholders, who are often distinct from the organizational-level stakeholders.

As GameDevCo transitioned from start-up mode to a full-fledged product company, most of the founders left the organization, and new senior leadership was brought in, creating the first set of misalignments between project-stakeholders and senior leadership. The recent acquisition of GameDevCo's by a global conglomerate has further resulted in misalignments with their corporate customers. At the root causes of the misalignments are: a limited understanding of GameDevCo's product (and by extension, the market they serve), the lack of visibility into GameDevCo's processes, and corporate ownership of the high-level product roadmap. This transition also resulted in a more hierarchical organization structure, wherein more layers were inserted between the project teams involved in the product lifecycle (design, development and sustainment), and their customers. At the organizational level, even within GameDevCo, there are misalignments between the various business units.

A case in point is the lack of alignment between the Infrastructure & Support (I&S) business unit, and the Developmental Studio (DS) that develops the games. The lack of alignment here arises from the absence of a structured release process for product upgrades – as a result I&S has to constantly juggle patches, and managing version control, straining an already limited staff. At the project-level, there is a strong alignment between the customers and developers in DS, as the developers have a strong passion for the game for which they developed the product. The misalignment with the architects/product owners is driven by shift in emphasis from 'new technology development' to 'business value'. As the head of DS noted for a product that has been in existence for almost six years,

We only wrote our first business story six months ago.

The lack of alignment between the developers/maintainers and the architects is driven by both the constant churn in the development process as well as the perception of the effectiveness of the architect/product owner. As one of the developers in the first generation product noted:

The people designing the new system created the mess in the first product – over the last two years, we have had to constantly redesign and refine the product while keeping it in service – getting it to the maturity we have

The challenges faced by the maintainers, as similar to those faced by the I&S stakeholders – they lack sufficient information to effectively support the product. Since the adoption of SCRUM as the development methodology of choice three years ago, the rapid execution cycle time has led to poor fidelity artifacts (when they exist). These artifacts tend to decay rapidly, as the collective problem solving activities that are carried out to resolve issues/make design changes are not captured. As one developer noted,

We are sprinting, not doing SCRUM

The assessment of the stakeholder alignment within GameDevCo is shown in Table 2.They have pockets of strong alignment but lack of alignment of organizational level stakeholders, has had an adverse effect on the project-stakeholders.

4 Employee Empowerment

The concept of employee empowerment appeared as part of the management vernacular in the 1980's and rapidly became an often used, yet poorly understood concept. As Randolph noted in 2000 [21], *empowerment remains one of the most promising, yet mystifying, concepts in business.* As Conger and Kanugo note [22], there are multiple constructs that are associated with empowerment: a relational construct at the organizational level in which the principle source of power that an actor has over the organization arises from the actor's ability to provide some performance or resource that is valued by the organization or the actors ability to cope with important organizational contingencies or problems (as defined by Pfeffer [23]); a relational construct at the interpersonal level often implies the leader/manager sharing power with their subordinates [24]; and a motivation construct wherein empowerment focuses on enabling employees.

Table 2. Stakeholder Alignment at GameDevCo

Stakeholder Alignment	Organization-Level Stakeholders		
	Senior Leadership	Customers	
		External	Internal
GameDevCo	○	○	○

Project-Level Stakeholders				
Customer	Architect	Developer	Maintainer	User
●	○	●	○	○
Legend ● – Strong Alignment ○- Weak Alignment				

As Ayoma points out [25], one of the goals of agile/iterative software development is to transition from a culture of enforcement to a culture of empowerment. In looking at the cultural assumptions underlying the adoption of CMM-based improvement efforts (which are predominantly used in organizations that use plan-based development approaches), Ngwenyama and Neilsen [1] note that empowerment is implicit in organizations that have a developmental orientation in which the focus is on human development. From the perspective of enabling software organizations in creating enterprise-level agility, employee empowerment is essential. People act as the primary sensemaking mechanism in software organizations, and an un-empowered employee cannot effectively be a part of creating or contributing to enterprise agility. Empowerment is seen as a critical factor in enabling software process improvements [26],but actually creating empowered employees is difficult [27]. Quinn and Spreitzer [28] synthesized the literature on employee empowerment to identify four characteristics of empowered people to be: a sense of self-determination (free to choose how they do their work); a sense of meaning (care about what they are doing); a sense of competence (confident about their ability to do the work); and a sense of impact (ability to influence their work unit). These four characteristics provide the structure that we can use to better understand employee empowerment in GameDevCo, as shown in Table 3.

While the adoption of agile processes in general increases the autonomy of the employee, and thereby enables them to make better project-level decisions, GameDevCo's adoption of SCRUM as the development approach of choice has brought some interesting organizational issues to the forefront – primary among them, the absence of training for contract employees, and lack of an effective tool to support the rapid development cycle time. Their explosive growth over the last four years, has resulted in GameDevCo's heavy reliance on contract employees to ensure project progress – however, these employees are not trained in the SCRUM process. As one contract employee noted:

I have been here for a year, and I still don't know what SCRUM is.

This problem is further exacerbated within the organization, as there is a wide-ranging understanding of what SCRUM is, and how it should be implemented. As a result, the nature of the day-to-day development activities is strongly influenced by the project manager (or scrum master). As one senior project manager noted:

We spend more time talking about what SCRUM is, and what it should be, rather than focusing on what it should be doing for us.

This internal process diversity, coupled with a lack of an effective governance structure has resulted in churning on the part of the employees when it comes to the development process. Another challenge that impacts their ability to do execute day-to-day work, is the lack of a requirements management tool that supports the rapid development cycle time. The corporate mandated tool, was designed to support plan-based development, and is too cumbersome to support effective situational awareness. As a workaround to the tool, development teams started using a wiki-based tool, however, even there, the artifacts decay rapidly. From the perspectives of meaning and competence, GameDevCo has world class technology competence, and across the board, the development teams are passionate about the game for which they developed the product. GameDevCo's growth has led to a greater focus on the business of selling games, and a perception of dilution on technology aspects of game development. This growth has also created a more hierarchical organization that has resulted in functional silos, as opposed to the cross-disciplinary, cross-lifecycle teams that existed in the initial phases of their growth. This problem has been further exacerbated by the move to a common office in a major city, which has exposed their technical staff to a larger, and in some cases more rewarding job market. From an impact perspective, GameDevCo's approach of using independent teams to developing the next generation product, and sustain existing products, has resulted in frustration for the team that sustains the current product. As one of the team members involved in sustainment noted:

There is always a ghost in the room – we don't really know what our role is when the next-generation product will finally be fielded.

Table 3. Employee Empowerment at GameDevCo

Empowerment Characteristics	**Self Determination**	**Meaning**	**Competence**	**Impact**
GameDevCo	●	●	●/○	○
Legend	**● – strong alignment**		**○ – weak alignment**	

5 Group and Organizational Learning

Software development is in many ways, one of the purest forms of knowledge work. There have been multiple studies that emphasize the importance of managing knowledge [29,30] and supporting organization learning in software development [31-34]. For understanding the strategies at GameDevCo, we use the three lenses of individual learning, group learning, and organizational learning (as shown in Table 4).

Table 4. Learning Strategies at GameDevCo

Learning Strategies	**Individual**	**Group**	**Organizational**
GameDevCo	Process Training	Informal Communities of Practice, Standup meetings, Retrospectives	University Collaboration

GameDevCo's focuses primarily on creating mechanisms to support individual and group learning. Given their primary challenge of standardizing to the SCRUM process, every member of the permanent staff went through basic training in the process. As was noted earlier, this training was not provided to contract staff, a significant portion of their total human capital. To support their transition efforts, they retained an external consultant to mentor their SCRUM masters, however, the consultant left prior to institutionalizing the knowledge, resulting in significant variation in the development process. As one SCRUM master noted:

We spend more time arguing about what the scrum books say about the process, rather than in trying to figure out what the process is trying to do for us

Table 5. Analysis Governance Strategies at GameDevCo

Governance Analysis	**Strategic Governance**			**Project Management**	
	Business Model Alignment	**Project Portfolio Management**	**Decision Rights**	**Policy/Process**	**Tool Support**
GameDevCo	○	○	○/●	○	○/●
Legend	**● – Strong practices observed**		**○ – Weak or no practices observed**		

The growth of the organization has resulted in a separation of game development and systems operation – further limiting the opportunities for individuals to learn from the debugging of server related problems and the creation of release packages. Group learning at GameDevCo currently occurs through the use of informal communities of practice that meet to share best practices and lessons learned. The challenge however is that these communities of practice do not meet on a regular basis due to schedule pressures. Another approach to fostering group learning is the use of retrospectives at the end of an iteration, which by definition are meant to provide the team with an opportunity to reflect on both the project and the process. These meetings however, have now become a pro-forma ritual. As one developer noted:

Our retrospect starts with the scrum master saying, What didn't work in this sprint? What can we improve in the process?... We look around and in 5 minutes we are done

One of the points that is often made when discussing organizations that adopt agile methods, is the large amount of tacit knowledge within the organization – i.e. organization that is resident in the heads of individuals. With GameDevCo, the challenges arise from the lack of organizational infrastructure to support learning, as well as the simple fact that the organization is executing too fast to learn.

6 Systems of Governance

The notion of governance in software organizations has received little explicit focus from the academic community until recently – a case in point being the First Workshop on Software Development Governance that was held as part of ICSE'08. The summary of the workshop identifies the perceptions of 21 participants on what they believe is part of governance process. We coded the raw data to extract four common themes of: Bridging Strategy and Execution; Project Portfolio Management; Allocation of Decision Rights; and Project Monitoring and Control. While the understanding of governance is skewed to the individual's perception based on their location in the organization and prior experience (in current and previous organizations), there is strong consensus that governance bridges strategic and operational aspects of software development, and it includes some aspect of project management. Our analysis builds around the same structure, addressing first the strategic aspect of governance, and then the day-to-day activity of project management.

GameDevCo is in the process of figuring out how governance should be carried out. Their original business model (which was focused on servicing individual clients), has since been expanded to services and infrastructure provision. Moreover, their interactions with corporate headquarters appear to be 'mandate-based', as opposed to collaborative tailoring. Their transition to SCRUM placed the onus of decision making on the development teams themselves however, the corporate process for project management is built around the traditional monitoring and control paradigm that relies heavily on stable technology roadmaps, and a stage-gated product development process. To further exacerbate the mismatch is the lack of a mapping between the roles specified in the corporate project management process, and the roles used within GameDevCo. As one senior project manager noted:

We have raised this lack of clarity on the roles multiple times, but there has been no action taken by corporate to resolve it.

Given that the corporate office does not really understand how GameDevCo's processes work, it becomes critical for them to have an effective project management system. The SCRUM methodology by definition comes with an implicit project management strategy that has a strong emphasis on self-governance, with the SCRUM masters (SMs) and Product Owners (POs) acting as the interface to senior leadership. As discussed earlier, there is still no consensus within GameDevCo on exactly what their implementation of SCRUM is. In addition to lack of clarity on the process itself, there is significant heterogeneity on the 'who' and 'how' aspects of governance as well. The reporting internal to project teams is done in terms of burn-down charts, but the translation to the larger enterprise is unclear. Similarly, there is significant overlap in the roles of the SMs, POs and Architects.

7 Discussion

The four organizational enablers emerged from a grounded theoretic analysis of interviews, observations, and archival data. The engaged scholarship approach [13], wherein we engaged the members of GameDevCo at multiple levels, allowed us gain a nuanced understanding of the challenges that they faced in adopting agile methods. The approach overall of starting with an engagement mindset, and combining multiple data collection methods, provides an effective means of carrying out firm level research. Furthermore, the results are trustworthy as triangulation was carried out at the data and theoretic levels; and both the study participants and other peers validated the results. The next step in the research will be to determine whether the findings apply to process improvement in non-agile contexts. From the standpoint of the practitioner, we emphasize the need for addressing both the mechanics and the dynamics of agile adoption. Most organizations implicitly assume that agile practices will automatically ensure stakeholder alignment and employee empowerment – GameDevCo is an illustrative example of that problem. In the absence of a holistic system, that enables learning, and implements strategic governance to enable stakeholder alignment and create employee empowerment, agile adoption will not translate to strategic agility, at best, it will enable production agility.

Acknowledgement

The first author would like to thank the members of GameDevCo, who generously shared their time, and their insights during the multiple cycles of data gathering and analysis This research was supported in part, by the Swedish Foundation for Strategic Research (SSF), through the PROGRESS center at Mälardalen University.

References

[1] Ngwenyama, O., Nielsen, P.A.: Competing values in software process improvement: an assumption analysis of CMM from an organizational culture perspective. IEEE Transactions on Engineering Management 50(1), 100–112 (2003)

[2] Teece, D.J., Pisano, G., Shuen, A.M.Y.: Dynamic Capabilities And Strategic Management. Strategic Management Journal 18(7), 509–533 (1997)

[3] Zollo, M., Winter, S.G.: Deliberate Learning and the Evolution of Dynamic Capabilities. Organization Science 13(3), 339–351 (2002)

[4] Ethiraj, S.K., Kale, P., Krishnan, M.S., et al.: Where do capabilities come from and how do they matter? A study in the software services industry. Strategic Management Journal 26(1), 25–45 (2005)

[5] Athreye, A.S.: The Indian software industry and its evolving service capability. Industrial and Corporate Change 14(3), 393–418 (2005)

[6] Ono, T.: Toyota Production System: Beyond Large-Scale Production. Productivity Press (1988)

[7] Spear, S., Bowen, H.K.: Decoding the DNA of the Toyota Production System. Harvard Business Review 77, 96–108 (1999)

[8] Sugimori, Y., Kusunoki, K., Cho, F., et al.: Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. International Journal of Production Research 15(6), 553–564 (1977)

[9] Liker, J.K., Meier, D.: Toyota Talent: Developing Your People the Toyota Way. McGraw-Hill, New York (2007)

[10] Morgan, J.M., Liker, J.K.: The Toyota Product Development System: Integrating People, Process, and Technology. Productivity Press (2006)

[11] Roth, G., Labedz, C.: Lean Enterprise Change Case Study: Rockwell Collins. MIT, Cambridge (2006)

[12] Heeks, R., Krishna, S., Nicholson, B., et al.: Synching or Sinking: Global Software Outsourcing Relationships (2001)

[13] Van de Ven, A.H.: Engaged Scholarship: A Guide for Organizational and Social Research. Oxford University Press, USA (2007)
[14] Boehm, B., Bose, P.: A collaborative spiral software process model based on Theory W, pp. 59–68
[15] Boehm, B.: Anchoring the software process. IEEE Software 13(4), 73–82 (1996)
[16] Clements, P., Garlan, D., Little, R., et al.: Documenting software architectures: views and beyond, pp. 740–741
[17] Damian, D.: Stakeholders in Global Requirements Engineering: Lessons Learned from Practice. IEEE SOFTWARE, 21–27 (2007)
[18] Damian, D.E., Zowghi, D.: The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization, pp. 319–328
[19] Beck, K.: Extreme programming explained: embrace change. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
[20] Boehm, B., Turner, R.: Using Risk to Balance Agile and Plan-Driven Methods. Computer, 57–66 (2003)
[21] Randolph, W.A.: Re-thinking empowerment: Why is it so hard to achieve? Organizational Dynamics 29(2), 94–107 (2000)
[22] Conger, J.A., Kanungo, R.N.: The empowerment process: integrating theory and practice. Academy of Management Review 13(3), 471–482 (1988)
[23] Pfeffer, J.: Organizations and Organization Theory, Cambridge, MA (1982)
[24] Burke, W.: Leadership as empowering others. Executive power, 51–77 (1986)
[25] Aoyama, M.: Web-based Agile software development. IEEE Software 15(6), 56–65 (1998)
[26] Dyba, T.: An Instrument for Measuring the Key Factors of Success in Software Process Improvement. Empirical Software Engineering 5(4), 357–390 (2000)
[27] Baddoo, N., Hall, T.: Motivators of Software Process Improvement: an analysis of practitioners' views. The Journal of Systems & Software 62(2), 85–96 (2002)
[28] Quinn, R., Spreitzer, G.: The road to empowerment: seven questions every leader should answer. Organisational Dynamics 26(2), 37–50 (1997)
[29] Ashforth, B.E.: The experience of powerlessness in organizations. Organizational Behavior and Human Decision Processes 43(2), 207–242 (1989)
[30] Rus, I., Lindvall, M.: Knowledge Management in Software Engineering. IEEE SOFTWARE, 26–38 (2002)
[31] Basili, V.R., Caldiera, G., Rombach, H.D.: Experience Factory. Encyclopedia of Software Engineering 1, 469–476 (1994)
[32] Ruhe, G., Bomarius, F.: Learning software organizations. Springer, New York (2000)
[33] Schneider, K., von Hunnius, J.P., Basili, V.R.: Experience in Implementing a Learning Software Organization. IEEE SOFTWARE, 46–49 (2002)
[34] Althoff, K.D., Bomarius, F., Tautz, C.: Knowledge Management for Building Learning Software Organizations. Information Systems Frontiers 2(3), 349–367 (2000)
[35] Dyba, T.: Improvisation in small software organizations. IEEE Software 17(5), 82–87 (2000)
[36] Sawyer, S., Guinan, P.J.: Software development: Processes and performance. IBM Systems Journal 37(4), 552–569 (1998)

Migrating Defect Management from Waterfall to Agile Software Development in a Large-Scale Multi-site Organization: A Case Study

Kirsi Korhonen

Nokia Siemens Networks, Hatanpäänvaltatie 30, 33100 Tampere, Finland
kirsi.korhonen@nsn.com

Abstract. Defect management practices are considered an important part of traditional software development. While embracing agile methods, software development organizations have realized that defects still do exist and they must be managed. Therefore defect management practices should be migrated as well, but current instructions for such a change are fragmented or incomplete. We studied three software development organizations to find out what are the main problems to consider in defect management when migrating from waterfall to agile. We identified five issues related to process, tools and metrics in a multi-site organization. This paper proposes action items to deal with these issues during the agile migration planning activities.

Keywords: defect management, agile adoption, distributed development.

1 Introduction

Changing the development model from traditional waterfall towards agile methods is a challenge to any organization [18], [19], [20]. When software development is also distributed, it brings new challenges as it seems to be in contradiction with one of the success factors of agile development – the development should preferably take place in one location, meaning that people are physically located in the same room [21].

Successful transformations from traditional to agile software development have been reported [17], which encourages organizations to take agile methods into use. In addition, frameworks [15], [16] have been designed to guide organizations in the migration process. These include lists of possible challenges for successful agile adoption, such as distributed development [19]. One part of the migration process is selecting appropriate agile methods, for example, taking suitable defect related metrics into use [15], [24]. Still further studies in the field are needed on how defect management practices should be revised when adopting agile methods, especially in a multi-site organization.

The goal of this study was to analyze the problems of defect management when a globally distributed organization was migrating from traditional software development to agile. This included finding answers to what changes, if any, are needed for defect related metrics, processes and fault reporting tools. We also wanted to find out what additional challenges arise from distributed development to the adoption of agile defect management.

In this research we studied two large and one medium-sized globally distributed telecommunications software development organizations, which migrated from traditional development to agile. The data was collected by analyzing the project documentation and defect data, and by interviewing the key persons.

The results indicate that defect management has to be carefully planned when a large-scale organization is migrating to agile. In particular, the following recommendations should be considered: 1. Specify faults to be reported. 2. Create practices for prioritizing between fault fixing and feature development during a sprint. 3. Evaluate the appropriateness of the current fault reporting tool and related practices. 4. Analyze what defect management metrics should be used in agile. 5. Evaluate the multi-site development impact on defect management. .

This paper is organized as follows: Section 2 provides background information, and the research setup is described in Section 3. The empirical results are presented in Section 4, and analyzed in Section 5. The paper is concluded with final remarks.

2 Background

In this section we first describe the traditional defect management process and related challenges in distributed software development. Next we discuss agile defect management, and adoption of agile methodologies.

2.1 Traditional Defect Management

The main goal of defect management is to increase the software quality by finding and fixing defects as early as possible. Fenton [11] describes software defects to be both failures in the required behavior of the system, and faults in the software product caused by a human error. In order to systematically investigate the defects and provide corrections for them, there must be information recorded for each defect in a reporting tool [11].

The basic defect management process includes defect prevention, defect discovery and resolution, defect causal analysis and process improvement [10]. There are also other models available e.g. [6], [7], and practical instructions for establishing a defect management process in an organization [9]. The metrics in use should be suitable for defect management [11], and there are examples from industry [7], [8], [13] showing that improving the defect management process improves software quality.

2.2 Multi-site Defect Management

Industrial studies report challenges in traditional software development in a multi-site environment, such as coordination difficulties [2] and delays in cross-site work compared to same site work [3]. Impacts on defect management can be, for example, delays in getting the fixes in time, or miscommunication of the importance of a certain defect. Proposed solutions include improving the tool usage [12], negotiation practices [5], implementing an organization-wide defect management process [10] and using quality metrics to support the defect management process [13].

2.3 Defect Management in Agile Software Development

One agile viewpoint proposes that defects are seen as waste [21]. Defects should be fixed as soon as they are found and therefore, reporting the defects with a tool should not even be required. When a Scrum [1] team is co-located, and the team members can work together in the same room next to a shared flipchart, this may not be an issue. However, in a distributed organization the teams are working in different physical locations, so there may be communication problems as well as cultural differences. These have brought up a need for more formal communication between teams than agile method proponents originally proposed [20].

Additionally, if the system is old, the legacy code base is likely to have an unknown number of defects. Therefore, a zero fault tolerance may not even be possible. Consequently, the faults need to be managed, and there are recent reports on fault quality metrics in agile development [22] and fault prediction models [23].

Research results [17], [20] suggest that agile methods, such as pair programming, automated tests and continuous integration, help to improve product quality and reduce the number of faults. However, these results also show that although the number of faults has decreased, faults still need to be handled formally.

2.4 Adopting Agile Methodologies

In this section we briefly review and discuss defect management topics from agile transformation studies. In these studies, defect management is not discussed as a separate topic, but guidelines and experiences are fragmented over several documents.

Success stories of adopting agile methodologies, such as Primavera [17], highlight the benefits of the transition to agile development, but also give examples of possible challenges. At Primavera the organization was able to provide their customers with higher quality release, but a defect management problem in the beginning was that the development features were prioritized over fault fixes by the scrum teams. In order to fix all the faults, they needed to dedicate one full sprint in the end just for fixing the bugs that were left unfixed earlier.

Cohn and Ford [19] describe common pitfalls in migrating to agile and effective approaches for making the change. According to the study, distributed development proved to be challenging, and should not be used during the first two or three months after initiating an agile process. Further, Misra et al. [16] mention distributed development on their list of challenges for successful agile transition.

In Primavera [17] the stakeholders would have required a better picture of the progress, and burn down charts alone were not seen as a sufficient source of information. Accordingly, Cohn and Ford [19] propose that management still requires progress status reports. A typical Scrum status report including a list of key dates and metrics, such as defect inflow, would be satisfactory for decision making [19].

There are also structured approaches available to guide organizations in taking the agile practices into use. Sidky and Arthur [15] introduce a 4-stage process framework which includes an assessment of organizational readiness for agile transformation and identifying the set of agile practices and tools to be taken into use. Also Nerur et al. [24] emphasize that tools play a critical role and organizations planning to adopt agile should invest in tools that support rapid iterative development. However, neither of these studies discusses migrating fault management tools or practices as a separate topic.

The research material introduced in this section proposes that when adapting to agile methodologies, topics such as distributed development, progress reporting and tools should be considered. As these are handled separately in the materials and not from defect management point of view, further research is needed with field studies to assess what kind of defect management practices are needed for adopting agile development.

3 Research Setup

This paper reports a case study [14] in which three multi-site organizations were studied in a real software development environment. This section gives an overview of the organizations, research methods and the data collection process.

3.1 Research Context

The organizations were selected in this research to represent different combinations of size and experience in applying agile practices. The context of the organizations was similar: experts working globally on different sites to develop software in the telecommunications domain. Organizations 1 and 2 were fully agile. The third organization had waterfall practices still in use on the main project level, while subprojects were agile. This mixed approach during transformation was used because the project schedule was tight and there had not been enough time to transform all the main project level criteria and processes to agile. Nevertheless it was still seen beneficial to have the subprojects in agile mode. Organization 3 results were collected from the whole project. Table 1 summarizes the main characteristics of the organizations.

Table 1. Main characteristics of the study organizations

	Organization 1	**Organization 2**	**Organization 3**
Size of the organization	Large (>150 people)	Medium (50-150 people)	Large (>150 people)
Involved countries	3	2	5
New or legacy software?	New	Legacy	Legacy
Years of agile practice	3	5	0.5
Number of agile releases	<5	<10	1

3.2 Research Methods and Data Collection

Triangulation [4] was used to get a good overview of the data by verifying the results with three different methods of data collection. We analyzed qualitative data in the project documentation and by interviewing key persons. Additionally, we collected quantitative fault data from each project.

The project documentation was first analyzed to get an overview of what defect management practices there were in place. The analyzed documentation included the project plan, project quality plan, fault related instructions and fault metric reports.

The participants (project manager, quality manager, system level testing manager and a scrum team member) were selected for the interview to represent different viewpoints of the project. Project manager (later referred to as PM) has the overall responsibility for the project; therefore he needs to have enough information on e.g. fault status for decision making. Quality manager (QM) defines the defect related quality metrics and guidelines for the project organization. The system level testing manager (SLT) is working with different teams over different sites in the organization and also needs to provide progress data to the project management. The scrum team member (STM) has an insight to the work within one team and how the defect management processes and tools support the daily activities.

Data was collected from individual people and focus groups with semi-structured interviews. The questions asked in the interviews were related to problems met during migration, problems caused by distributed development, reasons why some defect management practices worked or did not work, possible new practices developed for agile defect management, and possible requirements for the defect reporting tools. Due to the qualitative nature of the answers, no separate statistical analysis was done.

Quantitative fault data includes initial analysis of the collected defect records. Defect data was collected weekly from the fault reporting tool database during the latest software development project from each organization.

4 Empirical Results

In this section we present the results from the documentation study and interviews as well as the initial analysis of the defect data.

4.1 Qualitative Results

In this section the qualitative results are grouped into four categories. First we discuss about the changes to defect reporting process and defect metrics. Next we talk about the fault reporting tools, and finally discuss problems in agile defect management due to the multi-site organization. Based on the documentation review, basic defect management practices were used in all organizations (table 2).

Defect Management Process. In organization 3 there was a common misunderstanding on different levels that, while in agile, there is no need to report any faults at all with fault reporting tools. Similarly, the QM and STM of organization 1 commented that not all faults were reported to the tool. PMs and QMs wanted to have the faults reported to make the progress visible, but the teams were reluctant to report faults. This created a conflict between management expectations and Scrum team practices. To improve the situation, both organizations produced guidelines (by the QM) on what faults should be reported in the tool. These included e.g. faults between the teams and faults transferred from one sprint to another.

In organization 2, the fault reporting tool was used as a fault backlog. Fault classification in the tool to critical, major and minor gave priority for the faults, and a Scrum team member could work on any of the reported faults on their team's development responsibility. The SLT mentioned that this motivated people to report and fix faults, even though it was slightly confusing to have two separate backlogs, one for faults and one for features.

All organizations reported that it had been difficult to find the balance between feature priorities, program schedule and quality requirements. According to agile process definitions [1], [21], quality, and therefore fault fixing, is the first priority, and feature development comes second. In practice, implementing new features was seen as more important during the sprint than fixing minor bugs, so that bugs were transferred to the next sprint in all organizations. Organization 2 had taken this even further. First sprints were reserved mainly for development, and a few sprints in the end were reserved just for fault corrections to stabilize the system

Table 2. Main results from the documentation study

	Organization 1	**Organization 2**	**Organization 3**
Milestone critical defect criteria	No critical faults accepted. Project level target for critical and major faults, minor fault targets for components.	No critical and major faults accepted. Project level target for critical, major and minor faults	No critical faults accepted at milestone. Project level target for critical faults. Major and minor fault targets for subprojects.
Sprint exit defect criteria	No critical or major defects accepted.	No critical or major defects accepted.	No critical or major defects accepted.
Other defect related metrics	Fault closing speed	Cumulative open and closed faults	Fault closing speed Cumulative open faults
Status delivery	Once a week	Three times a week	Five times a week
Defect management process description.	Part of the quality management documentation.	Part of the fault reporting tool instructions.	Separate fault management process description.

Fault Reporting Tool. None of the organizations replaced their fault reporting tool with a new tool during agile migration. All organizations had company-wide reporting systems in place, and it was not possible to change it. Interviews revealed that using the existing tools helped the communication between the teams as the format of the data in the reporting tool was familiar.

However, there were proposals about how the tool should be improved to better fit the agile way of working. According to the TMs and SLTs, the tool should be more flexible than the current tool and it should be easier and faster to use when reporting a fault. According to the STM from organization 3, there were too many mandatory data fields for each fault. The SLT from organization 1 commented that some of the tool instructions were updated in response to changed information needs.

Metrics. According to the QMs, the number of open faults was the key metric in their organizations with two differences compared to the waterfall metric. Instead of measuring the number of open faults only at major milestones, in agile it was measured after each sprint. The second change was in the target level setting. In waterfall, the number of open faults was generally higher, and targets were defined for each subproject separately. In the agile mode, the expectation was that there would not be that many faults, so target levels could be set on project level. Organization-specific target settings had evolved along with gained experience (Table 2).

All organizations provided fault status reports weekly with a slight difference in delivery intervals (Table 2). In organization 3 the fault status report format was taken as such from the waterfall model. There were already further development ideas, for example, the STM suggested that follow-up should be feature-based.

Multi-Site Organization. A general comment from all organizations was that introducing agile did not bring any new multi-site problems to their defect management. The same problems still existed as with the waterfall development model: delays in communication and long response times in fixing and verifying the faults between the teams. The SLT from organization 2 mentioned a change that instant messaging was introduced to improve the communication between the teams.

4.2 Analysis of the Defect Data

Defect data was collected weekly directly from the fault reporting tools. Figure 1 presents the number of open defects over time by organizations. The measurement period was divided into three phases: development, final testing and fault fixing. The development phase consists of content development sprints. During the final testing phase the testing was done for the whole product, and the fault fixing phase contained fault correction and verification tasks. Each phase length was at least one sprint.

In organization 1 the graph of open defects has only small variation during development sprints. This is to be expected in agile software development, as the faults are to be fixed as soon as they are found [21]. The graph was giving a too positive message, and the high peek during final testing was not anticipated. To stabilize the system, an extra fault fixing sprint was scheduled.

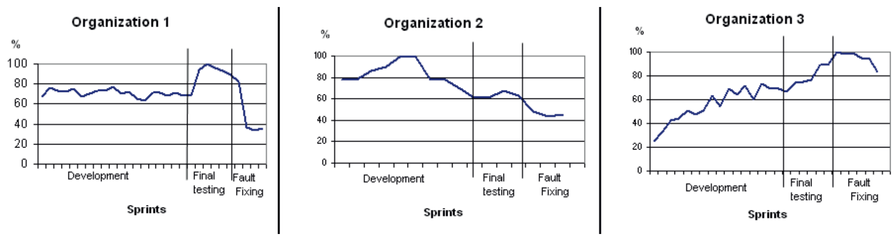


Fig. 1. Open faults index (% of max number of open faults) from Organizations 1, 2 and 3

In organization 2 the number of open faults was at highest during the development sprints and after that it steadily decreased sprint after sprint. The graph indicates that defects were reported, and the defect management practices, such as transferring minor bug fixes and pre-scheduling fault correction sprints, supported the work.

From the graph of organization 3 it can be seen that the agile transition did not yet have a major impact on the defect management, as the number of open faults was heavily increasing during the development sprints. This can be explained by inexperience with working in agile mode as the transition was still ongoing.

5 Discussion

The analysis of the results revealed several problems. Fault reporting and correction prioritization was not clear in the scrum teams. Metrics needed fine-tuning as well as the fault reporting tool practices and communication between the different locations.

If there is no agreed defect management process, it can cause problems [11]. We noticed that in defect management migration to agile, unclarity about which faults still need to be reported, can cause confusion. In consequence, communication between teams can suffer, defect data is unreliable and it cannot be used for reliable progress follow-up. Guidelines on what faults need to be reported help the situation. Additionally, the fault reporting tool can be introduced as a fault backlog.

Interviewees in our study reported that it was difficult to prioritize between the fault corrections and feature development during one sprint due to pressure from the different stakeholders. The same problem was also seen at Primavera [17]. However, our study revealed that a controlled way to manage this was to allow minor class fault fixes to be transferred to the next sprint to give more time for feature development. To handle these fault fixes properly and ensure the quality, one or more separate fault fixing sprints should be planned in the end to stabilize the software.

An agile adoption study [24] proposes to evaluate the appropriateness of the tools for agile. According to our study, this should cover not only the fault reporting tools but also the tool related practices, such as what data is filled in which field, what fields are mandatory and what kinds of reports are generated based on the data. It was considered useful that the tool was not changed in agile, because it made communication easier as everybody was already familiar with how to use it.

Our study showed that the defect metrics can be reused from the waterfall process, but they need to be modified to support the agile development. For example, the number of open defects should be followed up during each sprint and on the project level to get the overall picture. Also the target setting should be revised.

Table 3. Problem areas and proposed solutions

Problem area	Proposed solution practices
What faults need to be reported?	Create guidelines on what faults need to be reported. Introduce fault reporting tool as a fault backlog.
How to prioritize fault fixing and feature development during sprint?	Establish a common process how to prioritize between fault fixing and feature development. Allow minor fault fixes to be transferred to next sprint. Schedule fault fixing sprints.
Should the existing fault reporting tool be used?	Evaluate the appropriateness of the current tool and tool related practices. Create common guidelines on how to use the tool in agile.
What metrics to be used in agile?	Evaluate the metrics used in waterfall, decide which ones can be used in agile and redefine target levels. Follow up on sprint level.
How to manage distributed defect management?	If the organization is not familiar with distributed development, get familiar with agile first. If development is already done in multiple locations, evaluate the defect management communication practices.

Agile transformation guidelines [16],[19] mention that distributed development is a challenge in agile transformation. This might be the case also for defect management migration in organizations which are unfamiliar with distributed development. But according to our study, if an organization is already doing software development in a distributed environment, agile development as such does not bring any new multi-site challenges to defect management. In such cases the cultural differences, communication issues, and practices of software integration are usually sorted out already before the migration. However, timely communication is essential because of short development sprints, and needs special attention also in the agile working mode.

As an outcome of the analysis, five problem areas were identified. These problem areas and proposed action items to address them during the migration from waterfall to agile are presented in Table 3.

6 Conclusions

The contribution of this paper lies in providing empirical evidence on how adopting agile methods affects defect management. There are case examples in previous research of successfully adopting agile practices in organizations, and frameworks to guide in the transformation process. In this paper the agile adoption is analyzed from a defect management point of view by collecting experiences from three organizations about the effects of agile adoption on the fault management process, tools, metrics and multi-site challenges.

Based on the results, five problem areas were identified, and the proposed actions are respectively 1) create guidelines on what faults need to be reported, 2) evaluate the appropriateness of the fault reporting tool and related practices, 3) establish a common process how to prioritize between fault fixing and feature development, 4) select the metrics to be used in agile, and 5) make sure that communication practices are well established between distributed teams.

In future research we will focus on evaluating further the results of agile transformation measured by defect data and collect requirements for an agile fault reporting tool.

Acknowledgements. The author thanks Kai Koskimies, Pekka Abrahamsson, Hannu Korhonen, Erik Hiltunen and Maria Lahti for their comments and insights.

References

1. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall, Englewood Cliffs (2002)
2. Herbsleb, J.D., Grinter, R.E.: Splitting the organization and integrating the code: Conway's law revisited. In: Proceedings of International Conference on Software Engineering, pp. 85–95 (1999)
3. Herbsleb, J.D., Mockus, A., Finholt, T.A., Grinter, R.E.: Distance, dependencies, and delay in a global collaboration. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work, pp. 319–328. ACM Press, New York (2000)

4. Jick, T.D.: Mixing Qualitative and Quantitative Methods: Triangulation in Action. Administrative Science Quarterly 24(4), 602–611 (1979)
5. Sandusky, R.J., Gasser, L.: Negotiation and the coordination of information and activity in distributed software problem management. In: Proceedings of international ACM SIGGROUP conference on supporting group work, pp. 187–196. ACM Press, USA (2005)
6. Florac, W.: Software quality measurement a framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22 (1992)
7. Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P.: Experiences with defect prevention. IBM Syst. J. 29(1), 4–32 (1990)
8. Daskalantonakis, M.K.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. IEEE Transactions on Software Engineering 18(11), 998–1010 (1992)
9. Quality Assurance Institute: Establishing a Software Defect Management Process. Research report number 8 (1995)
10. Jäntti, M.: Difficulties in Establishing a defect management process: A Case Study. LNCS, pp. 142–150. Springer, Heidelberg (2006)
11. Fenton, N.E., Pfleeger, S.L.: Software Metrics, A rigorous and practical approach. PWS Publishing Company (1997)
12. Larsson, A.: Making sense of collaboration: the challenge of thinking together in global design teams. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, pp. 153–160. ACM Press, New York (2003)
13. Korhonen, K., Salo, O.: Exploring Quality Metrics to support Defect Management Process in Multi Site Organization – a Case Study. In: Proceedings of ISSRE (2008)
14. Yin, R.: Case Study Research: Design and Methods. Sage Publishing, Thousand Oaks (1994)
15. Sidky, A., Arthur, J.: A disciplined approach to adopting agile practices: the agile adoption framework. In: Innovations in Systems and Software Engineering, pp. 203–216. Springer, Heidelberg (2007)
16. Misra, S., Kumar, U., Kumar, V., Grant, G.: The organizational changes required and the challenges involved in adopting agile methodologies in traditional software development organizations. In: Digital Information Management, pp. 25–28 (2006)
17. Schatz, B., Abdelshafi, I.: Primavera Gets Agile: A Succesful Transition to Agile Development. IEEE Software 22, 36–42 (2005)
18. Lawrence, R., Yslas, B.: Three-way cultural change: Introducing agile with two non-agile companies and a non-agile methodology. In: Proceedings of AGILE Conference (2006)
19. Cohn, M., Ford, D.: Introducing an Agile Process to an Organization. Computer 36(6), 74–78 (2003)
20. Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., Kahkonen, T.: Agile software development in large organizations. Computer 37(12), 26–34 (2004)
21. Poppendieck, M., Poppendieck, T.: Lean Software development. Addison-Wesley, Reading (2007)
22. Concas, G., DiFrancesco, M., Marchesi, M., Quaresima, R., Pinna, S.: An agile development process and its assessment using quantitative object-oriented metrics. In: XP 9th international conference, Ireland, pp. 83–93 (2008)
23. Catal, C., Diri, B.: A fault prediction model with limited fault data to improve test process. In: Profes 9th international conference, Italy, pp. 244–257 (2008)
24. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to agile methodologies. Communications of the ACM 48(5) (2005)

Perceptive Agile Measurement: New Instruments for Quantitative Studies in the Pursuit of the Social-Psychological Effect of Agile Practices

Chaehan So and Wolfgang Scholl

Department of Organizational and Social Psychology
Institute of Psychology, Humboldt University
Rudower Chaussee 18, 12489 Berlin, Germany
chaehan.so@gmail.com, schollwo@cms.hu-berlin.de
http://tinyurl.com/agilestudy

Summary. Rising interest on social-psychological effects of agile practices necessitate the development of appropriate measurement instruments for future quantitative studies. This study has constructed such instruments for eight agile practices, namely iteration planning, iterative development, continuous integration and testing, stand-up meetings, customer access, customer acceptance tests, retrospectives and co-location.

The methodological approach followed the scale construction process elaborated in psychological research. We applied both qualitative methods for item generation, and quantitative methods for the analysis of reliability and factor structure (principal factor analysis) to evaluate critical psychometric dimensions.

Results in both qualitative and quantitative analyses indicated high psychometric quality of all newly constructed scales. The resulting measurement instruments are available in questionnaire form and ready to be used in future scientific research for quantitative analyses of social-psychological effects of agile practices.

Keywords: agile practices, measurement instruments, iteration planning, iterative approach, continuous integration, test-driven development, stand-up meetings, co-location, retrospectives, customer acceptance tests, customer access.

1 Introduction

Research has investigated agile methods since the late 1990s, while the underlying roots date back to the 1980s [1]. Growing interest in psychological aspects within the agile software domain has since revealed that there is more to agile methods than the technical and process issues, namely a vast area of human and social aspects. A number of studies have emerged investigating various social aspects in the complex relationship between agile practices and team interaction [2,3].

All these studies have applied qualitative research methods and found positive results of specific agile practices. Future research must therefore corroborate these qualitative findings using quantitative methods. In addition, the highly influential non-scientific consultant and practitioner literature on agile methods is abundant with claims about positive effects on social phenomena. The question remains whether these claims can be substantiated in quantitative studies. In pursuit of answering such research questions through quantitative analyses, this study aimed to develop appropriate measurement instruments.

2 Theoretical Framework

For quantitative studies on social-psychological effects, it is crucial that the measurement of agile practices is not effected by technical tools, but from the angle of *perception*. In psychology, perception is defined as the outcome of human information processing involving encoding, storage, retention, information retrieval, and judgment. The theoretical underpinnings of this approach derive from the fundamental psychological mechanism that individuals' behavior and choices are more influenced by their perceptions of situations and contexts than by the objective factual situation [4]. In many cases of real-life work contexts, discrepancies can occur between the technical reality (e.g. implementation quality of agile practices) and the corresponding individuals' perceptions. These discrepancies are caused by the fact that perceptions are prone to distortions through various cognitive and emotional bias effects, e.g. *selective attention* or *confirmation bias* (i.e. the tendency to ignore facts contradictory to prior judgement) [5].

Our selection of agile practices is by no means an attempt to present an exhaustive coverage of agile methodology on the whole, but to establish a representative set of agile practices commonly used in the field. Beyond coverage of agile values and principles as delineated in the *Agile Manifesto* [6], this set was intended to provide a solid basis for the process of scale construction and validation of agile practices as described in the methods section.

Consequently, we chose to pursue the following agile practices (their core aspects denoted in parentheses): *Iteration planning* (participation of all team members), *iterative development* (short iterations, time-boxing, working software), *continuous integration & testing* (continuous integration, test-driven development), *stand-up meetings* (short, regular, focused), *customer acceptance tests* (frequent, requirements verification by the customer), *customer access* (ease of contact to the customer, useful feedback), *retrospectives* (identification and implementation of improvement points), *co-location* (degree of physical proximity).

3 Method

The framework to develop measurement instruments for agile practices in this study is derived from the methodology of *scale construction* that has been developed and refined by psychological research for the last 100 years.

Psychological scale construction follows several qualitative methods for item generation and quantitative methods for the validation of established scales [7]. If the proposed new instruments are going to be used for future meaningful analyses in contexts of social interaction, they must undergo the scrutiny of reliability analysis and validation. In this study, we chose *internal consistency* as the main measure for reliability and analyzed *convergent validity* and *discriminant validity* (often summarized as *construct validity*) by explorative factor analyses.

3.1 Sample

In this study, team members (N = 227) of 55 software development teams applying agile practices across industries were tested. The sample consisted of project and product release teams, spread over 15 countries in America (Brazil, USA), Europe (Austria, Belgium, Finland, France, Germany, Italy, UK, Switzerland) and Asia (Bangladesh, India, Korea, New Zealand, Russia). Participants were male (n = 204) and female (n = 23). Work experience in agile software development projects was majorly distributed in the small range (62.0% with up to 1 year), considerable less in the middle range (21.9% with 1-2 years) and high range (16.1% with 3 years or more).

Acquisition of the participants followed five sources: Personal contacts of the author from eight years of professional experience in the software domain (mainly agile methods), contacts through researchers in the agile and organizational psychology domain, online newsgroups for agile methods, acquisition through an online business network (Xing), and an article published in the IT journal OBJEKTspektrum [8] which holds the highest subscription rate across German-speaking countries.

3.2 Procedure

The field study was conducted through a web-based questionnaire. Participants were assured of total anonymity by a formal data privacy policy signed by the research institute.

The agile practices' part of the main questionnaire was developed in three subsequent phases, followed by the final study:

Item Generation. The main qualitative measures we applied during the item generation and validation processes consisted of several structured and unstructured interviews with experts in the field through email, phone and face-to-face conversations. The starting point of the questionnaire development was the work of William Krebs who created the *Shodan Adherence Survey* [9] which was further elaborated by Layman et. al [10]. The scales of this survey were composed of multiple items; these items were not evaluated individually but only by a single-item for the whole scale. Hence, the corresponding psychometric quality indicators were not available. Consequently, we decided to use the shodan survey items solely as a start discussion basis for the solicited experts to generate a new item pool basis. We applied qualitative procedures of item validation involving that new items were added and existing items were discarded, modified, extended or shortened, based on the expert feedback. Finally, three of the 84 items[1] of the Shodan Adherence Survey were retained in our final questionnaire version (see appendix) comprising 48 items.

Pretest 1. The first prelimenary study was conducted 19-26 June 2008 with 37 participants from 13 teams. Participants were mainly recruited from the network of the author's personal contacts. Feedback on the survey items, composed of questions, misunderstandings and clarifications, were discussed and the result integrated into the subsequent revision.

Pretest 2. The second prelimenary study was conducted 15-24 October 2008 with 44 participants. Participants were recruited through announcement in six major online newsgroups for agile software development.

Final Study. The final study was conducted between 26 October 2008 and 31 January 2009. From the pool of 260 formally invited persons, a total of 87.3% (N=227) in 55 software development teams completed the main questionnaire containing 43 items about agile practices. The co-location scale (5 items) was answered in an additional short questionnaire on project data by the technical project managers or scrum masters.

The data collection started with 107 individuals in 21 teams. After the first week, more than 700 practioners of agile methods, enlisted in the online business network Xing (www.xing.com), were contacted with information about this research study and a request to participate. From this channel, a total of 120 additional participants could be invited, and another 33 participants from further inquiries within the personal business network.

3.3 Optimizing Variance Explained

In pretest 1, a 5-point Likert scale format ranging from 1 (not at all) to 5 (totally) was used. To increase *variance explained* in the final study according to Lozano [11], the number of response categories was extended by two additional points to a 7-point Likert scale. In consideration of participant feedback received from pretest 1, the response format was transformed from an extent to a frequency scale ranging from 1 (never) to 7 (always), with the exception of the co-location scale which maintained the 5-point ordinal scale ranging from -2 (different time zones) to +2 (same room).

[1] Item cit8 retained in original version and items acctest2, standup2 in adapted versions.

3.4 Content Validity

All items generated from the initial version underwent thorough review by six experts in agile methodology and by active practitioners in software engineering throughout the first three research stages item generation, pretest 1 and pretest 2. The experts were asked for each agile practice scale to comment on a number of aspects including '*do the question correctly reflect the main characteristics of this agile practice?*', '*would you delete any question?*', '*how would you modify a question to improve its validity?*', '*would you add any question?*'. In addition, we solicited open feedback by means of a comment functionality in the used survey tool. This feedback was collected and systematically compared with prior feedback, merged and integrated into the next questionnaire version. This whole process was repeated several times during each stage until feedback from all involved experts converged to satisfaction with content validity.

3.5 Reliability

A scale is regarded to be reliable if there is little variance that is specific to particular items [12]. The most wide-spread measure used in psychological research for reliability is *internal consistency* which is equivalent to the average of all possible combinations of *split-half reliability* (i.e. splitting the scale by every possible combination of items according to the Spearman-Brown formula). The most commonly used statistic for internal consistency is *Cronbach* α. Acceptable values of α coefficients are generally regarded in the range of 0.75 and above; yet it is important to emphasize that the factor structure must be considered as well because the differential diagnostic value decreases the more dimensions the scale incorporates.

3.6 Factor Structure

When testing several psychological constructs simultaneously, a psychological test must ensure that these constructs can be measured as distinctly separate from each other. The according standard validation method during scale construction is *explorative factor analysis*. The goal of this method is to test for a clear *factor structure* which is defined by *convergent validity* (items for one construct load on the same factor and with high factor loadings determined by the marking factor) and *discriminant validity* (items for different constructs load on different factors and without double loadings).

In the field, agile practices are mostly applied at the same time; we thus expected a certain correlation of the respective scales and accounted for it by applying the oblique rotation technique *direct quartimin*[2] which allows maximum possible correlation in the solution as described by Gorsuch [13].

For the extraction technique, we chose to apply *principal factor analysis* (PFA) in favor of *principal components analysis* (PCA) because PFA, as argued by Tabachnick and Fidell [14, p.633–636], is designed for studies which hypothesize specific underlying constructs behind the empirical data.

[2] Direct quartimin corresponds to direct oblimin with a gamma value of zero.

In order to verify whether the right number of factors (8) was extracted, we used several criteria: First, the *scree* criterion indicated extraction until the 7th factor; yet we extracted an 8th factor because of a corresponding eigenvalue of 1.45 which clearly fulfilled the *Kaiser* criterion[3]. The most important of all applied criteria was that factor extraction should essentially be guided by *interpretability* of factors. In our case, all factors extracted corresponded precisely to agile practices modeled in our data. One aggregate scale (continuous integration & testing) spread its two subscales (continuous integration and test-driven development) over two different factors and thus revealed to be a two-dimensional construct. The aggregation of these two subdimensions was justified by the high reliability of the aggregated scale (0.88).

4 Results

The following section presents the results of the statistical analysis methods applied, namely reliability analysis and explorative factor analysis employing principal factor extraction and direct oblimin rotation.

4.1 Reliability Analysis

First, we analyzed internal consistency coefficients on the individual level. In order to improve our level of confidence, we tested internal consistency additionally on the group level by using aggregated individual item scores for the calculation of alpha coefficients[4].

Results (table 1) show consistently high Cronbach α coefficients for all scales (ranging between 0.78 and 0.93 on the individual level). These high reliability values could be replicated on the group level, showing even slightly higher values, ranging between 0.82 and 0.95.

4.2 Principal Factor Analysis

The analysis of the factor structure following the principal factor analysis approach showed a distinctly clear factor structure for all scales[5] (table 2).

The factor correlation structure revealed relatively low correlations among components (24 of 28 correlations below .31, 4 correlations in the range between .33 and .39). The highest correlation (.39) was between the scales retrospectives and customer acceptance tests which both share occurrence after iteration end.

[3] The Kaiser criterion specifies to extract all factors with eigenvalues above 1.

[4] Exception: The co-location scale was exclusively answered by project managers resp. scrum masters, hence no aggregated group value could be calculated.

[5] Co-location scale excluded from principal factor analysis because this scale was evaluated solely by project managers resp. scrum masters.

Table 1. Reliability Analysis on Individual and Group Level

Scale	# items	Cronbach α Indiv. Level	Cronbach α Group Level
Iteration Planning	7	0,79	0,85
Iterative Development	7	0,79	0,83
Cont. Integr. & Testing	9	0,88	0,93
Co-Location	5	0,78	n/a
Stand-up Meetings	5	0,79	0,82
Customer Access	4	0,93	0,93
Customer Acceptance Tests	5	0,87	0,91
Retrospectives	6	0,91	0,95

Table 2. Pattern Matrix of Principal Factor Analysis

	Factor							
	1	2	3	4	5	6	7	8
retrosp2	.87							
retrosp3	.86							
retrosp1	.77							
retrosp4	.75							
retrosp5	.68							
freqretr	.56							
cit5		.88						
cit9		.85						
cit6		.84						
cit8		.81						
cit7		.64						
cit4		.58						
access3			.92					
access1			.84					
access2			.84					
access4			.80					
acctest4				.84				
acctest1				.83				
acctest2				.77				
acctest3				.72				
freqcat				.40				
plan2					.65			
plan3					.65			
plan1					.59			
plan6					.47			
plan4					.46			
plan5					.37			
plan7								
standup2						.74		
standup1						.69		
standup3						.67		
standup5						.59		
standup4						.55		
cit3							.84	
cit2							.57	
cit1							.39	
iterat5								.64
iterat7								.54
iterat1								.52
iterat2								.44
iterat6								.40
iterat4								.37
iterat3								.36

Note: Factor loadings below 0.3 (<9% of variance explained) are omitted in this table

5 Discussion

The reliability analysis has shown that all scales possess distinctly high internal consistency to be categorized as highly reliable according to most psychological methodologists [12]. Moreover, since Cronbach α coefficients are consistently in the range of 0.8 and above, the scales can also be used for *causal analysis*. This is true not only for the analysis of causal structures with single or multiple regression, but also for the analytically more sophisticated *structural equation models*. Furthermore, we can assume that the high sample size of over 200 individuals yields relatively stable correlation parameters through a subject to item ratio of above 5:1, considerably higher than the recommendation of 2:1 by Kline [15].

The factor structure using principal factor analysis showed a clear discrimination between all scales and subscales spread over eight factors, also referred to as *simple structure*. The revealed simple structure can be regarded as a result of subsequent scale modifications based on the factor analyses after the preliminary studies. For example, items of the scale iteration planning were modified to reach unidimensionality of the scale after encountering high multicollinearity in the first pretest.

Future research should test the constructed scales of this study in other samples since results of reliability and factor analyses are always somewhat prone to sample dependency. These studies should apply *confirmative factor analysis* in order to validate the factor structure. The challenge to be taken consists of the requirements of high sample size for the mathematical method applied (structural equation modeling), and the main model fit criterion (Chi^2) which must be minimized for optimal model fit but has a positive linear dependency on sample size. The solution path of choice for this dilemma consists in reducing the number of estimation parameters to obtain more stable estimations. Item parceling techniques appear to be promising approaches in this direction.

In light of the aforementioned considerations, the decisive question emerges: How *replicable* are the results of this study? To answer this question, high reliablity and a clear factor structure are good indicators of replicability, but we must also look at *generalizability* – in this latter aspect, it seems safe to say that this study fulfills high requirements due to high sample size and because the sample was recruited across 15 countries, and furthermore varies from small companies to international corporations and across a wide spectrum of industries. This variability clearly distinguishes this study from previous research which mainly focused analysis on a single team or on several teams within the same company; we can thus expect a comparatively higher generalizability of this study's outcome. Yet it will be indispensable to test the new scales in future quantitative studies with confirmative factor analysis to reach an optimum level of confidence.

References

1. Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J.: New directions on agile methods: a comparative analysis. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering, Washington, DC, USA, pp. 244–254. IEEE Computer Society, Los Alamitos (2003)
2. Robinson, H., Sharp, H.: The social side of technical practices. In: Baumeister, H., Marchesi, M., Holcombe, W.M.L., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 100–108. Springer, Heidelberg (2005)
3. Whitworth, E., Biddle, R.: The social nature of agile teams. In: Proceedings AGILE 2007, Washington, DC, USA, pp. 26–36. IEEE Computer Society, Los Alamitos (2007)
4. Thomas, J., Clark, S., Gioia, D.: Strategic sensemaking and organizational performance: Linkages among scanning, interpretation, action, and outcomes. Academy of Management Journal 36(2), 239–270 (1993)
5. Nickerson, R.S.: Confirmation bias: A ubiquitous phenomenon in many guises. Review of General Psychology 2(2), 175–220 (1998)
6. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for agile software development (Online) (August 2001)
7. Giles, D.C.: Advanced Research Methods in Psychology, Routledge, East Sussex, UK, New York, USA (2002)
8. So, C.: Teamwork in agilen Softwareteams (Teamwork in agile software development teams). OBJEKTspektrum Schwerpunkt: Kosten und Nutzen von Vorgehensmodellen (OBJEKTspektrum Focus: Cost and Benefit of Software Development Processes) (1), 10 (2009)
9. Williams, L., Layman, L., Krebs, W.: Extreme programming evaluation framework for object-oriented languages version 1.4. Technical Report TR-2004-18, North Carolina State University, Department of Computer Science, Raleigh, NC (June 2004)
10. Layman, L., Williams, L., Cunningham, L.: Motivations and measurements in an agile case study. Journal of Systems Architecture: the EUROMICRO Journal 52(11), 654–667 (2006)
11. Lozano, L.M., García-Cueto, E., Muñiz, J.: Effect of the number of response categories on the reliability and validity of rating scales. Methodology: European Journal of Research Methods for the Behavioral and Social Sciences 4(2), 73–79 (2008)
12. Cortina, J.M.: What is coefficient alpha? an examination of theory and applications. Journal of Applied Psychology 78(1), 98–104 (1993)
13. Gorsuch, R.L.: Factor Analysis. Lawrence Erlbaum Associates, Hillsdale (1983)
14. Tabachnick, B.G., Fidell, L.S.: Using Multivariate Statistics, 5th edn. Allyn & Bacon, Needham Heights (2006)
15. Kline, P.: An Easy Guide to Factor Analysis. Routledge, London (1993)

Appendix: Perceptive Agile Measurement (PAM) Scales

Item Name	Item Wording	Corrected Item-Total Correlation	Cronbach α if Item Deleted
Scale: Iteration Planning			
plan1	All members of the technical team actively participated during iteration planning meetings.	0.58	0.75
plan2	All technical team members took part in defining the effort estimates for requirements of the current iteration.	0.62	0.75
plan3	When effort estimates differed, the technical team members discussed their underlying assumption.	0.59	0.75
plan4	All concerns from team members about reaching the iteration goals were considered.	0.57	0.76
plan5	The effort estimates for the iteration scope items were modified only by the technical team members.	0.41	0.78
plan6	Each developer signed up for tasks on a completely voluntary basis.	0.55	0.76
plan7	The customer picked the priority of the requirements in the iteration plan.	0.40	0.79
Scale: Iterative Development			
iterat1	We implemented our code in short iterations.	0.53	0.76
iterat2	The team rather reduced the scope than delayed the deadline.	0.47	0.77
iterat3	When the scope could not be implemented due to constraints, the team held active discussions on re-prioritization with the customer on what to finish within the iteration.	0.45	0.78
iterat4	We kept the iteration deadlines.	0.50	0.77
iterat5	At the end of an iteration, we delivered a potentially shippable product.	0.62	0.74
iterat6	The software delivered at iteration end always met quality requirements of production code.	0.54	0.76
iterat7	Working software was the primary measure for project progress.	0.54	0.76
Scale: Continuous Integration & Testing			
cit1	The team integrated continuously.	0.50	0.88
cit2	Developers had the most recent version of code available.	0.40	0.88
cit3	Code was checked in quickly to avoid code synchronization/integration hassles...	0.39	0.88
cit4	The implemented code was written to pass the test case.	0.61	0.87
cit5	New code was written with unit tests covering its main functionality.	0.79	0.85
cit6	Automated unit tests sufficiently covered all critical parts of the production code.	0.78	0.85
cit7	For detecting bugs, test reports from automated unit tests were systematically used to capture the bugs.	0.64	0.87
cit8	All unit tests were run and passed when a task was finished and before checking in and integrating.	0.73	0.86
cit9	There were enough unit tests and automated system tests to allow developers to safely change any code.	0.80	0.85

Item Name	Item Wording	Corrected Item-Total Correlation	Cronbach α if Item Deleted
	Scale: Stand-Up Meetings		
standup1	Stand up meetings were extremely short (max. 15 minutes).	0.55	0.80
standup2	Stand up meetings were to the point, focusing only on what had been done and needed to be done on that day.	0.73	0.74
standup3	All relevant technical issues or organizational impediments came up in the stand up meetings.	0.64	0.77
standup4	Stand up meetings provided the quickest way to notify other team members about problems.	0.57	0.79
standup5	When people reported problems in the stand up meetigs, team members offered to help instantly.	0.58	0.79
	Scale: Customer Access		
access1	The customer was reachable.	0.84	0.90
access2	The developers could contact the customer directly or through a customer contact person without any bureaucratical hurdles.	0.82	0.91
access3	The developers had responses from the customer in a timely manner.	0.88	0.89
access4	The feedback from the customer was clear and clarified his requirements or open issues to the developers.	0.80	0.92
	Scale: Customer Acceptance Tests		
freqcat	How often did you apply customer acceptance tests?	0.48	0.88
acctest1	A requirement was not regarded as finished until its acceptance tests (with the customer) had passed.	0.75	0.82
acctest2	Customer acceptance tests were used as the ultimate way to verify system functionality and customer requirements.	0.77	0.82
acctest3	The customer provided a comprehensive set of test criteria for customer acceptance.	0.67	0.84
acctest4	The customer focused primarily on customer acceptance tests to determine what had been accomplished at the end of an iteration.	0.79	0.81
	Scale: Retrospectives		
freqretr	How often did you apply retrospectives?	0.64	0.90
retrosp1	All team members actively participated in gathering lessons learned in the retrospectives.	0.75	0.89
retrosp2	The retrospectives helped us become aware of what we did well in the past iteration/s.	0.82	0.88
retrosp3	The retrospectives helped us become aware of what we should improve in the upcoming iteration/s.	0.84	0.88
retrosp4	In the retrospectives (or shortly afterwards), we systematically assigned all important points for improvement to responsible individuals.	0.72	0.89
retrosp5	Our team followed up intensively on the progress of each improvement point elaborated in a retrospective.	0.70	0.90
	Scale: Co-Location		
coloc1	Developers were located majorly in ...	.52	.76
coloc2	All members of the technical team (including QA engineers, db admins) were located in ...	.67	.71
coloc3	Requirements engineers were located with developers in ...	.66	.71
coloc4	The project/release manager worked with the developers in ...	.50	.76
coloc5	The customer was located with the developers in ...	.49	.77

A Survey of Perceptions on Knowledge Management Schools in Agile and Traditional Software Development Environments

Finn Olav Bjørnson[1] and Torgeir Dingsøyr[2]

[1] SINTEF Fisheries and Aquaculture
[2] SINTEF ICT
NO-7465 Trondheim, Norway
{finn.o.bjornson,torgeir.dingsoyr}@sintef.no

Abstract. Knowledge management is important for software development, whether this is done using traditional or agile methods. In an exploratory survey on how agile and traditional companies view current practice and future importance of knowledge management approaches, we found that agile companies seem to be more satisfied with their knowledge management approaches when compared to traditional companies. Further, when comparing perceptions between small and medium sized companies, we found that medium sized companies are more satisfied with their knowledge management approaches than small companies.

Keywords: Agile software development, knowledge management, scrum, extreme programming, empirical software engineering, survey.

1 Introduction

Software development is knowledge intensive work, where employee knowledge is critical for project success. In a competitive environment with constant technological changes, it is a challenge to make room for reflection and analysis in the daily work to stimulate learning.

Knowledge management is an area that has received much attention in software engineering, for example through special issues [1], books [2], and survey articles [3, 4]. Most of the approaches are, however, focused on the needs of large enterprises. They can afford an infrastructure for managing knowledge, like processes and tools for rigorous analysis of experience from past projects. Thus, much of the existing work relates to codifying knowledge in experience repositories, making overviews of competence areas of employees and establishing separate departments in charge of experience transfer ("experience factories", [5]).

Many small companies have recently turned to agile development. This differs from traditional development methods in many ways, including how knowledge is managed. In agile processes, knowledge sharing happens through interaction – programmers share knowledge by working together and through close relations to the customers. Examples of specific practices aimed at knowledge transfer are pair programming in

Extreme Programming, reflection on practice in daily meetings in Scrum, close cooperation with the customer through the "planning game" and reflection on practice in sprint retrospectives in Scrum. Nerur et al. [6] state that knowledge management is vital to organizations, and that traditional software development has relied primarily on explicit knowledge available in documents. Agile development, however, relies on tacit knowledge in the heads of the team members.

The context of the research reported here is an action research project focusing on knowledge management for small and medium-size companies. In order to focus the research in the project, we were interested in *identifying how agile and traditional companies view current practice and future importance of knowledge management approaches*. We think this is important to see if the difference identified by researchers can be seen in practice, and in order to make sure that research is relevant for practice.

In the following, we will first present previous research on knowledge management in software engineering and in particular addressing agile software development. We then describe how we designed our survey on knowledge management approaches in Section 3. We present results in Section 4, and finally conclude in Section 5.

2 Knowledge Management and Agile Development

Knowledge management is a broad and interdisciplinary field. To systematize the work in the field, Earl [7] has classified work on knowledge management into schools (see Table 1). The schools are broadly categorised as "technocratic", "economic" and "behavioural". The technocratic schools focus on information technology or management in supporting employees in knowledge-work. The economic school focuses on how knowledge assets relates to income in organisations, and the behavioural schools focus on orchestrating knowledge sharing in organisations.

Table 1. Earl's schools of knowledge management

	School	Focus	Aim	Unit
Technocratic	Systems	Technology	Knowledge bases	Domain Enterprise
	Cartographic	Maps	Knowledge directories	Activity
	Engineering	Processes	Knowledge flows	Know-how
Economic	Commercial	Income	Knowledge assets	
Behavioural	Organizational	Networks	Knowledge pooling	Communities
	Spatial	Space	Knowledge exchange	Place
	Strategic	Mindset	Knowledge capabilities	Business

In the following, we describe five of the schools that we see as most relevant for companies or departments developing software: The systems school focuses on codifying knowledge in bases or repositories. The cartographic school makes knowledgeable people in an organisation accessible to each other for advice, consultation, or knowledge exchange through skills management systems. The engineering school focuses on processes, and making descriptions of them available for employees. The organisational school focuses on describing the use of organisational structures (networks) to share or pool knowledge. The spatial school focuses on designing office space to foster knowledge sharing.

In a systematic review of empirical studies of knowledge management in software engineering up until August 2006 [3], we found that most studies concentrated on the engineering, systems and organizational schools of knowledge management. However, in agile software development, many practices such as using information radiators and co-locating teams relate to the spatial school, and practices like scrum of scrums relate to the organizational school.

In a systematic review of empirical studies of agile software development, Dybå and Dingsøyr [8] identified two studies focusing on knowledge management: Bahli and Zeid [9] examined knowledge sharing in an Extreme Programming project and a traditional project. They found that when the Extreme Programming model was used, the creation of tacit knowledge improved as a result of frequent contacts. Hilkka *et al.* [10] studied two development organizations using methods similar to agile development, and underline the importance of skilled team members with solid domain knowledge: "without these kinds of persons, the chosen approach would probably have little possibility to succeed". The difference between traditional and agile knowledge management, is further the focus of Melnik and Maurer [11]. They discuss the role of conversation and social interaction in effective knowledge sharing in an agile process. They suggest that the focus on pure codification is the principal reason that traditional teams fail to share knowledge effectively. These studies underline the importance of knowledge management for agile development, and acknowledge the tacit nature of the knowledge needed for software engineering.

Parts of the works in the engineering school are relevant to agile development. In particular the practice of conducting project retrospectives has become increasingly popular due to agile development. This is a phenomenon that has attracted interest from research, but in software engineering, mainly on various approaches to conduct retrospectives or post mortem reviews [3].

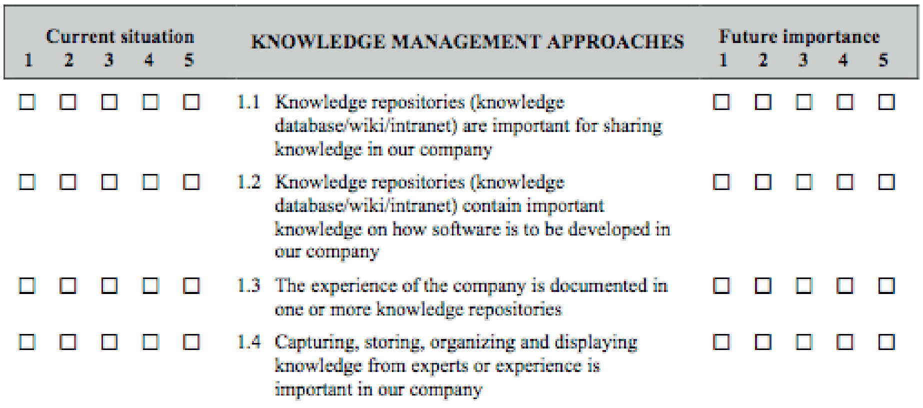
Another stream of research on knowledge management for agile development environments is works on communities of practice. This work would fit in the organizational school in Earl´s framework. The core idea in communities of practice is to get people who are working on similar tasks to share knowledge. This can for example be people who want to develop their skills on testing or project management who form a forum across projects. Holz and Maurer [12] present a system to support knowledge management in distributed teams. Kähkönen [13], explains how agile development works in multi-team settings using communities of practice theory.

Also, we find some work related to the spatial school. Sharp et al. [14] use theory of distributed cognition to analyze informal information flow between developers in an Extreme Programming project, discussing both the use of information radiators (or "walls") and how the development team is situated in their physical environment.

Desouza [15] discuss how to facilitate tacit knowledge exchange, and report from a study of a game room in a software company, which was reported to increase the exchange of project-based tacit knowledge.

3 Research Method

In order to identify current practice and future importance of knowledge management approaches within small and medium-size companies in a project in Belgium, Cyprus and Norway, we designed a participative assessment, inspired by Dybå and Moe [16]. The questionnaire was to be completed by a representative on behalf of the software development department in all 15 companies. The representatives were participating in an improvement project, and were typically responsible for a software development department, or working with quality assurance. The questionnaire contained one part with background information on the company and development method used, and one part on knowledge management approaches. For the knowledge management approaches, we used Earls schools of knowledge management, with four questions to determine the current situations and future importance for what we saw as the five most relevant schools: Systems, Cartographic, Engineering, Organizational and Spatial. An example question is shown in Fig. 1.



Current situation					KNOWLEDGE MANAGEMENT APPROACHES		Future importance				
1	2	3	4	5			1	2	3	4	5
☐	☐	☐	☐	☐	1.1	Knowledge repositories (knowledge database/wiki/intranet) are important for sharing knowledge in our company	☐	☐	☐	☐	☐
☐	☐	☐	☐	☐	1.2	Knowledge repositories (knowledge database/wiki/intranet) contain important knowledge on how software is to be developed in our company	☐	☐	☐	☐	☐
☐	☐	☐	☐	☐	1.3	The experience of the company is documented in one or more knowledge repositories	☐	☐	☐	☐	☐
☐	☐	☐	☐	☐	1.4	Capturing, storing, organizing and displaying knowledge from experts or experience is important in our company	☐	☐	☐	☐	☐

Fig. 1. Four questions from the questionnaire to determine current practice and future importance of the systems school

For each question, the respondents would indicate if they (1) strongly disagreed, (2) disagreed, (3) neither agreed nor disagreed, (4) agreed or (5) strongly agreed. Four questions would together form a score between 1 and 5, both for the current and future situation for each school.

The questionnaire was reviewed by researchers and practitioners before distribution, and completed by 3 companies in Belgium, 6 in Cyprus and 6 in Norway, in total 15.

The answers were recorded in a spreadsheet, and because of the small sample, we did not use statistical analysis, but analyzed the results using plots and tables as shown in the next Section.

4 Results

We now describe the data sources from our survey and the major results from our analysis.

From Table 2 we see that the companies ranged in size from seven to 250 employees. Since our main focus is on the knowledge sharing of developers we chose this field as the major classifier for size. Thus for this paper, small companies refers to companies with 2-18 developers, and medium companies range from 51-160 developers. With respect to development methods, the survey consisted of 8 questions to determine the current development methods and how the participants viewed the future degree of use of these methods in their company. We categorized self defined agile processes and scrum as agile methods, while unspecified self defined, waterfall and the Rational Unified Process was classified as traditional development methods. Where both waterfall and scrum was in use, we based our classification on the degree of use of the respective methods.

Table 2. Company background and classifications

#	Total size	Developers	Size category	Development method(s)	Development category
1	80	65	Medium	Waterfall and scrum	Agile
2	130	51	Medium	Scrum	Agile
3	250	70	Medium	Waterfall and scrum	Agile
4	200	160	Medium	Waterfall	Traditional
5	7	3	Small	Self defined	Traditional
6	20	15	Small	Waterfall	Traditional
7	25	8	Small	Waterfall	Traditional
8	10	2	Small	Waterfall	Traditional
9	15	5	Small	RUP	Traditional
10	25	6	Small	RUP	Traditional
11	14	5	Small	Self defined evolutionary	Traditional
12	11	6	Small	Scrum	Agile
13	28	10	Small	Waterfall and scrum	Agile
14	20	6	Small	Scrum	Agile
15	60	18	Small	RUP and Scrum	Agile

We performed several comparisons of the different variables, including country, development method and company size. The most interesting results from an agile point of view, we believe are the comparison of development method and size, which we report in this paper.

Table 3 provides an overview of the current status of the schools, the future expected importance and the gap between them, for companies using agile development methods and companies using traditional development methods. Thus, the gap is an indication of how important the companies believe the different schools are in order to improve their knowledge management practices. The gap size is illustrated in Fig. 2.

Table 3. Comparison of Agile and Traditional development methods

		Schools				
		Systems	**Cartographic**	**Engineering**	**Organizational**	**Spatial**
Agile	**Current**	3,57	3,04	3,21	3,18	3,79
	Future	4,36	3,46	4,25	3,82	4,04
	Gap	0,79	0,43	1,04	0,64	0,25
Traditional	**Current**	3,53	2,50	3,69	3,13	3,22
	Future	4,81	3,47	4,66	3,97	3,69
	Gap	1,28	0,97	0,97	0,84	0,47

As we can see from Fig. 2 the overall trend is that companies using agile development methods seem to be more content with their knowledge sharing (i.e. lower gap). The only exception is in the engineering school, where companies using agile methods have a slightly higher gap than the traditional.

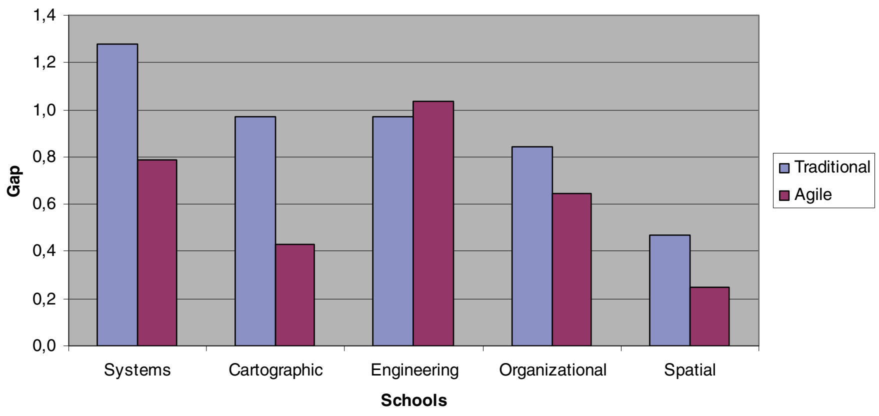


Fig. 2. Gap comparison between traditional and agile development methods

Going through each school in Table 3, we see that: The systems school start out with nearly the same current level, but traditional see a larger potential in developing this school. The cartographic school starts out with different current levels where agile is more satisfied, and both directions want to increase the level to nearly the same level. For the engineering school, both directions want to increase the school about the same, but with different starting points we again see that traditional puts more emphasis on this school. The organizational school has a close starting point and are also close in future importance, meaning both directions put about equal emphasis on this school. For the spatial school, the current status for agile is already above traditional, and the future importance of this school for traditional will still put it below the current status of agile. This means that agile puts more emphasis on the spatial school even though the gap for agile development is lower than the gap for traditional.

Table 4 provides the same overview as Table 3 but with the focus on small versus medium sized companies instead. The gap difference is illustrated in Fig. 3. As we can see from Fig. 3 the overall trend is that small companies feel a stronger need to improve all of the schools when compared with the larger companies. They especially perceive a need to improve the systems school, implying that larger companies usually have some sort of knowledge base in place, while smaller companies do not.

Table 4. Comparison of Small and Medium sized companies

		Schools				
		Systems	**Cartographic**	**Engineering**	**Organizational**	**Spatial**
Small	**Current**	3,27	2,58	3,45	3,20	3,32
	Future	4,57	3,31	4,48	4,02	3,73
	Gap	1,30	0,72	1,02	0,82	0,40
Medium	**Current**	4,31	3,25	3,50	3,00	3,94
	Future	4,69	3,94	4,44	3,56	4,19
	Gap	0,38	0,69	0,94	0,56	0,25

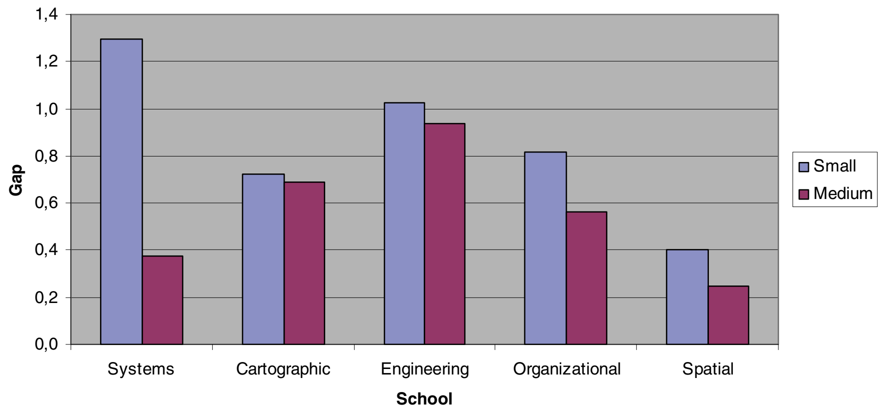


Fig. 3. Gap comparison between small and medium sized companies

Although the gap is similar in the cartographic school, the difference in current levels indicate that larger companies see a clearer need for the cartographic school. The engineering school is also remarkably similar in both current and future importance, indicating that both small and medium sized companies place equal importance on this school. Regarding the organizational school, we see that smaller companies place a higher importance here, comparing the future importance for this school it might seem like they overestimate how important it should be. Comparing spatial, we see that medium size companies place a far higher value on this both in current and future importance.

5 Conclusion and Further Work

We have conducted a survey amongst 15 small and medium-size software companies in Belgium, Cyprus and Norway in order to identify how agile and traditional companies view current practice and future importance of knowledge management approaches [3].

In previous work we have theorized that companies using traditional development methods might benefit more from the technocratic schools of knowledge management, while those using agile might benefit more from the more behavioural schools.

This impression is partly supported by this survey. We found that the systems and engineering school in particular are seen as having more potential by traditional companies when compared to agile. The spatial school was to a larger degree perceived to be adapted by agile than by traditional companies, but both directions seem to put the same emphasis on the organizational school, with agile companies having come further in the perceived implementation of this school. It is important to note however, that while agile companies place a high importance on the spatial school, the companies perceive that they are already using this school to a large degree, and as such the potential for improvement is perceived as low. Another important finding when comparing traditional development methods to agile is that companies doing agile development were overall more satisfied with their knowledge management than companies doing traditional development.

By comparing companies with respect to size, one school proved to have a large difference in gaps. There was a clear sentiment among small companies that the most potential was in the systems school. The big difference between small and medium sized companies here can be seen from the current score, where medium companies already express a high satisfactory level of implementation of this school. This indicates that as a company starts growing beyond small size they often put systems in place to ensure that knowledge is codified. Another important finding between small and medium sized companies is that smaller companies express a higher potential for improvement in all schools, indicating that companies that have grown past small size usually have a higher level of implemented knowledge management initiatives.

Our previous theory have been that agile companies will benefit more from behavioural knowledge management techniques than the technocratic techniques, a finding partly supported by the industrial actors through this survey. However as can also be seen from this survey, the agile companies themselves believe that the most potential for improvement lies within the technocratic schools. This might be explained through the fact that agile methods put emphasis on the behavioural schools, and so the need for improvement in these are seen as low compared to the schools that are not stressed by the agile methods. The EXTRA project will seek to further research this issue when trying out new knowledge management techniques in companies. This will involve trying our lightweight techniques in the engineering and systems schools, like descriptions on how to do project retrospectives, organizing project information and designing and using knowledge repositories.

In interpreting the results from this survey we are aware that the results are from a small subset of, and not a random sample of the relevant companies in general. Thus, we cannot generalize beyond our sample. Some immediate concerns would relate to the highly skewed sample in proportion of medium and small companies. In particular the fact that only one company is medium and traditional, thus making cross inferences between development methods and size difficult.

However, we see some of the findings as interesting enough to report since they partly support previous research and our impression from working with the industry. Future work in this area would be to conduct a survey on a larger random sample to see if the current indications hold in general.

Acknowledgements. The EXTRA project is initiated and coordinated by the Norwegian ICT association ICT Norway. The project is partly funded by Innovation Norway. We are also grateful to project partners CETIC and CITEA who collaborated in designing the questionnaire and collecting answers from companies in Belgium and Cyprus.

References

1. Lindvall, M., Rus, I.: Knowledge Management in Software Engineering. IEEE Software 19(3), 26–38 (2002)
2. Aurum, A., Jeffery, R., Wohlin, C., Handzic, M.: Managing Software Engineering Knowledge. Springer, Berlin (2003)
3. Bjørnson, F.O., Dingsøyr, T.: Knowledge Management in Software Engineering: A Systematic Review of Studied Concepts and Research Methods Used. Information and Software Technology 50(11), 1055–1168 (2008)
4. Dingsøyr, T., Conradi, R.: A Survey of Case Studies of the Use of Knowledge Management in Software Engineering. International Journal of Software Engineering and Knowledge Engineering 12(4), 391–414 (2002)
5. Basili, V.R., Caldiera, G., Rombach, H.D.: The Experience Factory. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, pp. 469–476. John Wiley, Chichester (1994)
6. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to agile methodologies. Communications of the ACM 48(5), 72–78 (2005)
7. Earl, M.: Knowledge Management Strategies: Towards a Taxonomy. Journal of Management Information Systems 18(1) (2001)
8. Dybå, T., Dingsøyr, T.: Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology 50, 833–859 (2008)
9. Bahli, B., Zeid, E.S.A.: The role of knowledge creation in adopting extreme programming model: an empirical study. In: ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society, pp. 75–87 (2005)
10. Hilkka, M.-R., Tuure, T., Matti, R.: Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. Journal of Database Management 16(4), 41–61 (2005)

11. Melnik, G., Maurer, F.: Direct verbal communication as a catalyst of agile knowledge sharing. In: 2nd Agile Development Conference (ADC 2004), Salt Lake City, UT, pp. 21–31 (2004)
12. Holz, H., Maurer, F.: Knowledge management support for distributed agile software processes. In: Henninger, S., Maurer, F. (eds.) LSO 2003. LNCS, vol. 2640, pp. 60–80. Springer, Heidelberg (2003)
13. Kähkönen, T.: Agile methods for large organizations - building communities of practice. In: Agile Development Conference, pp. 2–10 (2004)
14. Sharp, H., Robinson, H., Segal, J., Furniss, D.: The role of story cards and the wall in XP teams: a distributed cognition perspective. In: Agile Conference, Minneapolis, MN, p. 11 (2006)
15. Desouza, K.C.: Facilitating tacit knowledge exchange. Communications of the ACM 46(6), 85–88 (2003)
16. Dybå, T., Moe, N.B.: Rethinking the Concept of Software Process Assessment. In: European Software Process Improvement Conference (EuroSPI), Pori, Finland (1999)

Empowering Students and the Community through Agile Software Development Service-Learning

Joseph T. Chao[1] and Jennifer K. Brown[2]

[1] Department of Computer Science
[2] Department of English
Bowling Green State University, Bowling Green, Ohio 43403, USA
{jchao,jkbrown}@bgsu.edu

Abstract. This paper describes an approach to service-learning in the software engineering classroom that involves a central clearinghouse and maintenance center for service-learning project requests, use of Agile methods, and collaboration with a technical communication course. The paper describes the benefits and drawbacks to service-learning in a software engineering course, rationale behind using Agile, the course layout, specifics of the collaboration, the final feedback of the community partners and students involved, and a discussion of lessons learned.

Keywords: Teaching Agile Methods, Agile Software Development, Software Engineering Education, Pedagogy, Service-learning, Active Learning, Real-world Project.

1 Introduction

Service-learning has been a pedagogical approach used by many in software engineering courses to provide students with a real-world approach to learning software-development skills [1, 2, 3, 4]. Bringle and Hatcher [5] define service-learning as a "course-based, credit bearing educational experience in which students (a) participate in an organized service activity that meets identified community needs, and (b) reflect on the service activity in such a way as to gain further understanding of curricular content, a broader appreciation of the discipline, and an enhanced sense of personal values and civic responsibility." This service-learning approach to software engineering courses contrasts with the traditional approach in which students learn software-development skills by working in a lab on instructor-provided projects. Though this approach does provide valuable learning experiences, it does not provide the experience of working with a client to develop a software solution that meets the task-specific needs of the client, as the service-learning approach does.

Some argue that the service-learning approach to software engineering courses can prove advantageous to computer science departments that incorporate it, and it can prove advantageous to the entire computer science discipline [6, 7]. It has been suggested that the service-learning approach may attract better-performing students to the computer science discipline, which has been perceived with less interest by potential students in recent years [8].

1.1 Challenges to Implementing Service-Learning in Software Engineering Courses

Though the service-learning approach to software engineering courses has been embraced by many, instructors are often reluctant to incorporate service-learning into their courses because of the inherent drawbacks. Incorporating service-learning projects into a software engineering course requires extra time and organization from the instructor (e.g., soliciting non-profit organizations to serve as community partners, ensuring the partners are satisfied with their collaboration with the students), as well as some method of providing maintenance and technical support once the students who developed the software systems have completed the course.

Another challenge to incorporating service-learning projects into software engineering courses is the time limitations of a typical semester in an educational institution. Most students do not have any prior knowledge in software development before enrolling in the course, and typical semesters are limited to 16 weeks, which provides little time to teach students the building blocks they need to develop quality software systems. Moreover, a typical semester provides little time for students to meet with their teams, develop the software, meet with clients, and write documentation to accompany the software.

1.2 A New Approach to Service-Learning in Software Engineering Courses

Aware of the benefits and drawbacks of service-learning in software engineering courses, one instructor implemented a new approach in fall 2008 that mediated the main challenges of service-learning in a software engineering course. This approach required three steps:

1. Creation of the Agile Software Factory: The Agile Software Factory is a program within the Department of Computer science serves as a clearinghouse for software-development requests from non-profit organizations and continues to provide maintenance and technical support for completed software systems.
2. Collaboration with a technical communication class: The students in the technical communication class developed the documentation for the software, which freed more time for the software engineering students to develop the software and also provided both classes the opportunity to work within a real-world industry scenario.
3. Teaching and incorporation of Agile methods: Using Agile methods to teach software engineering allowed for better product development in a short time period.

2 Implementing Service-Learning in the Software Engineering Course

The three steps mentioned above for implementing a service-learning approach in the software engineering course are discussed in further detail in this section.

2.1 Involvement of the Agile Software Factory

The first step in incorporating the service-learning approach in the software engineering course was to develop the Agile Software Factory [9]. Founded in 2008, the Agile Software Factory (ASF) was begun with a grant from the Agile Alliance and sponsorship from the Computer Science department in which the ASF is housed, as well as sponsorship from the university's IT department.

The Agile Software Factory actively locates non-profit organizations that need software solutions developed, and it also receives unsolicited software requests from non-profit organizations. Once the ASF receives a software request, it evaluates its feasibility as a service-learning project for students in the software engineering courses. Once students complete the software system, the ASF provides ongoing maintenance and technical support services for the non-profit organization (i.e., the community partner). This reduces the burden on the instructor to provide maintenance for software systems developed in the courses taught by that instructor.

In addition to providing and supporting service-learning projects for software engineering courses, the ASF offers part-time student employment for undergraduate students and independent projects for graduate students.

In the fall of 2008, the ASF located six service-learning projects from community partners. These projects were assigned to teams of six to ten students, who were mostly undergraduate seniors and first-year graduate students.

2.2 Collaboration with the Technical Communication Class

The computer science instructor then collaborated with a technical communication instructor, who was teaching a senior-level undergraduate technical communication course focused on online documentation.

The instructors determined the basic responsibilities of the computer science students and the technical communication students and then assigned one or two (depending on the scope of the project and the size of the software engineering team) technical communication students to each of the software engineering teams. The software engineering students were responsible for producing programmer-oriented release notes that the technical communication students re-wrote for the intended clients. At the end of the semester, the technical communication students compiled and organized the release notes into a complete online help file for the user.

2.3 Use of Agile Methods

The instructor of the software engineering course decided upon teaching and using Agile methods for the service-learning project for three main reasons:

1. The growing use of Agile methods in industry
2. The focus on clients in the Agile approach
3. The iterative and incremental nature of Agile methods that would allow a working system to be built by the students in 16 weeks

Semester Schedule. Before the beginning of the semester, the instructor developed an iteration schedule that took advantage of Agile's focus on clients, as well as its iterative and incremental nature. The semester schedule was divided into five iterations, I0–I4, each consisting of three weeks (see Table 1).

Since most students were not familiar with software development methodologies, and were not aware of XP or other Agile methods, the first three weeks were designed to provide a quick overview of software engineering background and methods. Throughout the semester, several Agile methods and XP practices were also introduced in detail. The XP practices used in the course included the whole team, planning game, pair programming, coding standard, test-driven development, refactoring, and continuous integration.

Table 1. Semester schedule

Week(s)	Service-learning project task
1 & 2	Introduction to software engineering and the projects.
3	Formation of software development project teams and role assignments, as well as the software development and technical communication combined teams.
4	First customer meeting and Iteration 0 (I_0), which consisted of planning and a requirements analysis.
5	Second customer meeting to review requirements and project plan.
6 & 7	Work on I_1 and accompanying documentation.
8	Deliver I_1 (a working system) with accompanying release notes and updated project plan to client.
9	Work on I_2 and accompanying documentation.
10	Deliver I_2 (a working system) with accompanying release notes and updated project plan to client.
11	Work on I_3 and accompanying documentation.
12	Deliver I_3 (a working system) with accompanying release notes and updated project plan to client.
13–15	Work on I_4. Perform qualitative usability test.
16	Deliver final software product and accompanying documentation (as a compiled help file) to client.

Iteration Components. Iteration 0 consisted of project preparation, which included an initial meeting with the community partners, research on technologies, and preliminary project planning and estimation. The student teams were responsible for determining what they could commit to completing for the client by the end of the semester, based on client needs, team skills, and time limitations.

Iterations 1 through 4 required the student teams to complete user stories (system requirements) for each iteration and deliver a functioning product (a portion of the larger system) to the community partner at the end of the iteration. Each team arranged a meeting with their community partner at the end of each iteration to deliver the product and documentation.

The technical communication students were responsible for producing release notes to accompany the functioning product of each iteration. The release notes served as reference documentation for the community partner after the software engineering students installed the product and taught the partner how to use it.

Student Responsibilities. At the beginning of the semester, the students were provided the semester schedule shown above, developed by the instructor. Class sessions consisted of lectures that taught students the skills they would need to develop software. Only 3 class sessions were used for in-class work time, which meant students would have to work on the majority of their projects outside of class and schedule team meetings outside of class.

To complete each iteration, the teams were responsible for determining among themselves which team members would fulfill which roles (e.g., project manager, developer, tester). The teams were also responsible for determining who would complete which tasks on which days to ensure they completed the iterations by the deadlines and were meeting the needs of the community partner.

3 Outcomes

In fall semester of 2008, 46 students in two sections of the software engineering course were assigned to one of six teams to complete one of the six service-learning projects. By the end of the semester, the teams had completed the following projects:

- A victim case-tracking system
- An employee database system
- A service reporting system
- A service-learning information system
- A student-activity matching system
- An e-voting system

As with any new approach to teaching, and especially one that requires the coordination of multiple groups of people within a limited timeframe, this new approach to service-learning in the software engineering classroom posed a number of challenges (e.g., difficulty coordinating team member schedules, difficulty maintaining consistent communication with the technical communication students, difficulty completing iterations in a short period of time); however, the majority of the feedback from the community partners and students validated this approach and justified continuing it in future classes.

3.1 Student Feedback

At the end of the semester, the students agreed that they'd learned valuable skills from the service-learning project; in fact, 100% of the 46 students agreed that they had learned skills in the class that were applicable to the real world. Table 2 shows the results of an anonymous survey asking the students to rank their level of agreement with a series of statements about the course. Their responses indicate that most of the students valued the service-learning experience and the real-world skills they gained from their involvement in it.

In response to an open-ended question asking whether or not the students thought their participation in the course would improve their chances of landing a job after graduation, 34 of the 39 respondents to the question agreed that they indeed thought it

Table 2. Student survey results
(SA = strongly agree; A = agree; N = neutral; D = disagree; SD = strongly disagree)

Survey Statement	SA	A	N	D	SD
• The Agile methodology used in this class was a good approach for completing my project.	27	13	5	1	0
• My team project this semester is a good project to fulfill the purpose of this course. (45 respondents)	31	13	0	1	0
• I enjoyed working on the service-learning project in this course.	27	13	3	3	0
• I am satisfied with the project progress this semester.	14	24	3	4	1
• I understand the customer needs for the system.	24	21	0	1	0
• My team has produced a quality system that meets the customer needs.	18	24	3	0	1
• My team has worked with/interacted with the customer as professional service providers would.	15	22	7	2	0
• The communication between my team and the customer has been prompt and painless.	11	23	7	4	1
• My communication skills have improved this semester. (45 respondents)	15	24	4	2	0
• My teamwork skills have improved this semester.	16	25	4	1	0
• The skills I learned in this class are applicable to the real world.	35	11	0	0	0
• Our collaboration with the technical communicators on our team has been effective, purposeful, and useful.	12	13	17	2	2
• The technical communicators on my team have made a valuable contribution to our project.	14	17	10	4	1
• Collaboration with the technical communicators was a good idea and should be continued for this course in the future. (45 respondents)	15	17	11	0	2

would improve their chances of employment. In response to another open-ended question that asked "Name one thing you do not want to see changed about this course," 26 of the 36 question respondents listed the service-learning component. A number of students even mentioned the Agile approach itself. In fact, 31 of the 36 respondents to the question listed either (sometimes both) the service-learning or Agile components, or everything about the course.

Here are just a few of the students' actual comments, which reinforce their appreciation of the service-learning and Agile components of the course:

- "I thought I [could] never be a part of big projects like these."
- "I have learned that teamwork is more important than individual talent."
- "I think this class would not be nearly as effective without being a service-learning project. Going through the motions in a real-life situation is something that I think all [University] CS majors should be a part of."
- "I liked [using] the Agile method in a real-life project."

- "I had not heard of Agile software development before, but I am now glad I have. It is a great tool that I plan to use in the future."
- "I have gotten to see the other side of development, which is Agile. I also finally got to work on a project that matters in the real world."
- "Prior to this class, I was not sold on the idea of software development as a potential career. But after seeing all of the aspects of the process, I would love to go into development."
- "This is one of the most valuable courses in the entire CS dept."

3.2 Community Partner Feedback

All five of the community partners who responded to an anonymous survey (out of six who participated in the collaboration) reported that they enjoyed working with the team (see Table 3). Four of the five strongly agreed that they had received the same quality of software as professional developers would produce and that they would recommend the service-learning collaboration to other non-profit organizations.

Table 3. Community partner survey results
(SA = strongly agree; A = agree; N = neutral; D = disagree; SD = strongly disagree)

Survey Statement	SA	A	N	D	SD
• The student team has worked with/interacted with me as professional service providers would	1	3	0	1	0
• The communication between the team and me has been prompt and painless.	3	1	0	1	0
• I am satisfied with the project progress throughout the semester.	3	1	0	1	0
• The project was planned well and was carried out as planned.	4	0	0	1	0
• I enjoyed working with the team.	5	0	0	0	0
• The team has produced a useable system that meets my needs. (Only 4 responses)	3	0	0	1	0
• I feel that I received the same quality of software from this project as I would from professional developers.	4	0	0	1	0
• It was worthwhile for me to participate in this service-learning project.	4	0	1	0	0
• I will recommend this service-learning collabo ration to other potential organizations.	4	0	1	0	0

In response to the survey question "Name one thing you like about your finished product," the community partners responded with the following:

- "It was an excellent experience and we have a really useful system that we will implement in January. We would not be in a position to do this without the support of the team."
- "This is an excellent piece of software that will be useable by all employees of [organization]. It's fully functional, well thought out and exceeds [our] expectations."

- "(1) How easy it is to use. (2) The physical appearance is appealing to the eye."
- "It delivered the product that was promised. The product is not intimidating to the end users."
- "It is user friendly and it clearly shows the incredible progress the students made on this project."

The community partners were also asked to name one thing they disliked about their finished product, and answered with the following:

- "N/A"
- "We did not have time to adequately field test the system before the end of the project; but the team did agree to tidy up some final items."
- "Nothing"
- "The installation of the finished product should have been coordinated better on our end using our IT support contractor."
- "It does not allow us to do what we need as hoped. It doesn't allow for the complexity of reporting information needed; it doesn't have any dynamic areas that can grow with agency needs. All of these are things asked for and students committed to early on. It was clear by the last iteration, this was not possible. Despite this, the remarkable progress students made on this difficult group task is impressive."

When asked what they would like to see changed about the collaboration in the future, the community partners' responses fell into 3 basic categories:

- Clearer expectations at the beginning (e.g., required number of meetings, definitions of the documentation they would be receiving, structure of the collaboration)
- Allowing for more than one semester of work
- Improved student dynamics

When the community partners were asked what they would like to see unchanged about the collaboration, they responded with the following:

- "I would like everything to remain as it was. There was great communication between team members and the agency and the timeline was acceptable."
- "The focus on the final product and the focus on non-profit partners."
- "Everything."
- "I thought the assignment of clear responsibilities of the team members appeared to be a good group learning exercise and valuable experience in the real world."
- "The amazing talent of most of the students working to a collective good and the dedication of the instructor to the learning of the students."

Both sets of feedback provide valuable insight into what can be improved upon in the future of this service-learning collaboration, which is discussed in the Lessons Learned section below.

4 Lessons Learned

Based on the project results, and the feedback from both students and community partners, this service-learning software engineering approach was a success. In addition to achieving better management of all student projects, the instructor has gained insight in several other areas and will take the following steps in the future:

- Consider the availability of the clients when evaluating the suitability of the projects for the course. It would be valuable to have the clients sign an agreement for their time commitments. Ideally, the customer would be on-site, but because "customer on site" is not possible, the instructor will ensure the clients understand that customer availability is required before committing to the project.
- Schedule release meetings, if applicable, early in the semester. One problem encountered in the course was that one of the clients did not have time to meet and provide feedback at the end of each iteration.
- Use class time often, or at least at the end of each iteration, for retrospectives. The instructor has found that it is important to keep abreast of the progress of the projects and keep risk management in mind.
- Participate in customer meetings, especially the first meeting when finding out requirements is essential. In several meetings the instructor attended, students didn't always ask probing questions that would allow them to figure out the needs of the client, and they were often confused with system requirements. While it is not possible for the instructor to attend all meetings for all projects, it is crucial to attend the first meetings.
- Enforce some Agile practices such as pair programming and test-driven development. The instructor has found that pair programming not only improves the quality of the system, but also prevents students from procrastination.

With these new insights, the instructor will continue with this Agile service-learning approach to software engineering education. We are confident this approach is producing better-prepared software developers.

References

1. Liu, C.: Enriching Software Engineering Courses with Service-learning Projects and the Open-Source Approach. In: The 27th International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, May 15-21 (2005)
2. Poger, S., Bailie, F.: Student perspectives on a real world project. J. Comput. Small Coll. 21(6), 69–75 (2006)
3. Song, K.: Teaching software engineering through reallife projects to bridge school and industry. SIGCSE Bulletin 28(4), 59–64 (1996)
4. Tadayon, N.: Software engineering based on the team software process with a real world project. J. Comput. Small Coll. 19(4), 133–142 (2004)
5. Bringle, R.G., Hatcher, J.A.: A Service-Learning Curriculum for Faculty. Michigan Journal of Community Service Learning 2, 112–122 (1995)

6. Rosmaita, B.J.: Making Service Learning Accessible to Computer Scientists. In: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education Conference. SIGCSE 2007, Covington, Kentucky, USA (2007)
7. Purewal, T.S., Bennett, C., Maier, F.: Embracing the social relevance: computing, ethics and the community. In: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education Conference. SIGCSE 2007, Covington, Kentucky, USA (2007)
8. Carter, L.: Why students with an apparent aptitude for computer science don't choose to major in computer science. In: Proceedings of the 37th SIGCSE technical symposium on Computer science education. SIGCSE 2006, Houston, Texas, USA (2006)
9. Chao, J., Randles, M.: Agile Software Factory for Student Service Learning. In: The 22nd IEEE-CS Conference on Software Engineering Education and Training (CSEE&T 2009), Hyderabad, India, February 17-19 (2009)

Putting Agile Teamwork to the Test – An Preliminary Instrument for Empirically Assessing and Improving Agile Software Development

Nils Brede Moe[1], Torgeir Dingsøyr[1], and Emil A. Røyrvik[2]

[1] SINTEF ICT
[2] SINTEF Technology and Society
NO-7465 Trondheim, Norway
{nilsm,torgeird,emilr}@sintef.no

Abstract. Team organizing is a major way of assisting collaboration in knowledge intensive work such as software development, and is especially favored in agile approaches. Motivated by the challenge of transforming an organization from traditional *command-and-control* management to *collaborative self-managed teams,* we present an instrument that we argue addresses key concerns and characteristics of teamwork, and presents them along five dimensions that must be addressed when improving teamwork in agile software development. The dimensions are shared leadership, team orientation, redundancy, learning and autonomy. The instrument gives a radar plot of the status of the teamwork. We present empirical examples from using this instrument with three teams and briefly outline potential uses of the instrument.

Keywords: Agile software development, scrum, self-managed, self-organized, team radar, team effectiveness, empirical software engineering, case study.

1 Introduction

Various forms of collaboration between members is a foundational premise of agile software development methodologies such as Extreme Programming (XP) and Scrum [1]. Team organizing has been widely introduced as one major way of assisting collaboration in knowledge intensive work such as software development, and is especially favored in agile approaches. Agile methods accords primacy to uniqueness, ambiguity, complexity, and change, and the goal of optimization in traditional methods is being substituted by flexibility and responsiveness [2]. Team organizing is advocated as one major response to the organizational needs of agile approaches, especially in terms of work coordination.

In the traditional plan-driven approach, work is coordinated in a hierarchy involving a command-and-control style of management with clear separation of roles [2]. In the agile approach, ideally work is coordinated by the self-organizing team, where the team itself decide how work is coordinated [3]. Research on software development teams has found that team performance is linked with the effectiveness of teamwork coordination [4, 5].

How to coordinate and work effectively in a self-organizing team is not straightforward. A team following a plan-driven model often consists of independently focused self-managing professionals, and a transition to a self-managing team is probably one of the biggest challenges when introducing agile development [2]. Neither culture nor mind-sets of people are easily changed, making the move to agile methodologies all the more formidable for many organizations [3].

Motivated by the importance of team organizing in agile software development and the challenge of transforming an organization from traditional *command-and-control* management to *collaborative self-organizing teams,* this paper presents an instrument that we argue addresses key concerns and characteristics of teamwork, and presents them along five dimensions that must be addressed when improving teamwork in software development. We develop a framework for empirically describing these critical dimensions of teamwork, and motivated by a generic ambition combined with an action research approach [6] it is especially tailored towards supporting the introduction of, and change processes in, agile software development.

The rest of this paper is organized as follows: Section 2 gives an overview of research on teamwork in general and in software engineering and agile development. Section 3 defines what we believe are the five key characteristics of teamwork which is relevant for agile teams, as an instrument to measure and improve teamwork. Section 4 gives an empirical example from using this instrument on teams. Finally, we discuss the instrument and conclude on its usability in Section 5.

2 Research on Teamwork

Teamwork is a topic that has attracted research from several disciplines [7-9]. Consequently, there exists several definitions of teams, and the working definition we adopt is that a team is "a small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves mutually accountable" [10].

The issue of what processes and components comprise teamwork and how teamwork contributes to team effectiveness has received much attention [11, 12] Cross-functional, and self-organizing teams have been found to have high productivity [7] and increase the speed and accuracy of problem solving [13]. However, research indicates that the effects of such teams are highly situationally dependent and that the effects depend on factors such as the nature of the workforce and the nature of the organization [7, 8]. Literature suggests that self-management requires team orientation and backup behaviour [14], learning [14], adaptivity, and trust [2].

Agile software development represents a new approach for planning and managing software projects. The systematic review of empirical studies of agile software development [1] identify ethnographically informed studies by Sharp and Robinson that describe mature agile teams as having faith in their own abilities, showing respect and responsibility, having established trust, and preserving the quality of working life. Another research stream focusing on teamwork is Young et al. [15] work on personality characteristics for members of extreme programming development teams. In the recent literature on agile development, we further find studies of motivation and cohesion [16], individual characteristics and agile orientation [17], and motivational needs of extreme programming developers [18]. A study by Tessem and Maurer found high motivation and job satisfaction in a large agile team [19]. Acuña et al. [20] examined the perceptions on work procedures and practices within extreme programming teams. They found that high team vision preferences (or mental models) and high participative safety (degree of trust) perceptions of the team were related to improved software quality. Moe and Dingsøyr [21] found that Scrum had many of the mechanisms identified in the literature on team effectiveness in place, but found little support for team leadership and backup-behaviour.

The combined studies of teamwork within agile development indicates that agile methods are able to implement many of the characteristics of efficient teamwork from the general literature, like group cohesion, motivation, trust and adaptability. However, on some areas, which we will present in the next section, there is little overlap between the general literature and the agile literature.

3 An Instrument for Assessing and Improving Teamwork

In a large action research project comprising several companies using agile development methods, we found that teamwork was a major challenge. To improve teamwork, we needed a way to describe and diagnose the status of teams and a means of tracing and engaging with change processes.

For this purpose, we suggest the instrument described in the following, with five dimensions: shared leadership, team orientation, redundancy, learning and autonomy. The instrument is based on teamwork challenges identified in previous studies and on experience from action research with several agile teams. The instrument has been tried out on teams in two companies, as described in Section 4, and the feedback from these sessions has been integrated into the instrument.

The instrument consists of a series of open-ended questions for each of the five dimensions. Based on responses from a 20 minutes interview with all team members, the researchers discuss and give the team a score between 0 and 10 on each dimension. Finally, we plot the score of the team in a radar plot, and this is the basis of a feedback session with the team to discuss score and challenges.

The questions and scale is presented in the following, with a theoretical foundation for each of the dimensions.

3.1 Shared Leadership

In the literature of self-organizing and self-managing teams, it is claimed that the decision authority and leadership need to be shared [14, 22]. Pearce [23] argues that leadership should be rotated to the person with the key knowledge, skills, and abilities for the particular issues facing the team at any given moment. While the project manager should maintain the leadership for project management duties, team members should be allowed to lead when they possess the knowledge that needs to be shared or utilized during different phases of the project [24]. According to Hoegl and Parboteeah [25], all team members should also jointly share decision authority, rather than a centralized decision structure where one person makes all the decisions or a decentralized decision structure where all team members make decisions regarding their work individually and independently of other team members.

Insofar as possible, team leaders should select team members based on their technical, teamwork, and leadership skills. If shared leadership is to be developed, the right people must be on the team, and the teams need to be small enough [23]. The team leader should also be responsible for clarifying purpose, securing resources, articulate the vision and redesigning the team. He or she should also manage the boundaries of the team, articulate how the team will approach its task and function as a team (team process), and articulate trust and confidence in the team [23].

Questions: Is everyone involved in the decision-making process? Do team members make important decisions without consulting other team members? How is the team vision defined and presented? Is the team designed (and redesigned) according to its purpose?
Scale: 0 - Not shared leadership. Command and control by the team-leader, only a few takes part in the decision process. 10 - Shared leadership. The one that possess the knowledge leads, everyone is allowed to participate in the decision-making.

3.2 Team Orientation

Team orientation or collective orientation is often described as giving priority to team goals over individual goals [12]. Salas et al. [12] report on studies that show improved individual effort and performance, individual satisfaction, and increased team performance as a consequence on higher team orientation. Also, Salas et al. cite studies who found that teams with a team orientation more frequently consider teammate input when taking decisions.

Questions: How does the team take into account alternative suggestions in team discussions? How does the team value alternative suggestions? How do team members relate to the tasks of individuals? What kind of ownership do the team members have to the project?
Scale: 0 – Individual goals are more important than team goals, team members do not respect other team members´ behaviour. 10 – Team goals are more important than individual goals, team members respect other team members´ behaviour.

3.3 Redundancy

According to Morgan [14], any system with an ability to self-manage must have a degree of redundancy: a kind of excess capacity that can create room for innovation and development to occur. Members in a team need multiple skills so that they are able to perform (parts of) each other's jobs and substitute each other as circumstances demand. In this respect, socio-technical literature is concerned with "multiskilling" [26].

In psychologically oriented small-group research the concept of redundancy is often described as backup behavior [12]. If backup is to occur effectively, teammates need to be informed of each others' work in order to identify what type of assistance is required at a particular time [27]. Marks [27] identify three means of providing such backup behavior: (1) providing a teammate verbal feedback or coaching, (2) assisting a teammate behaviorally in carrying out a task, or (3) to complete a task for the team member when an overload is detected. Full redundancy is non-existent and the question is rather how much redundancy one seeks to build into the system.

Questions: How easy is it to complete someone else's task? If you are stuck; do you get help? Do you help others when they got problems? How are tasks allocated? If someone leaves the team; is it easy to substitute this person?
Scale: 0 - No redundancy. Team members are not able to give feedback, assist others with carrying out tasks, or complete task started by others. 10 - Full redundancy. Team members can replace other team members without extensive training.

3.4 Learning

Learning is important for teams for a number of reasons, including to develop shared mental models, in order to improve team performance, and also; studies of self-organizing teams show that this kind of organization requires a capacity for learning that allows operating norms and rules to change in relation to transformations in the wider environment [14]. One of the key inspirations for the Scrum method was the article on new product development by Takeuchi and Nonaka [28]. In this article, they describe "multilearning" as a key characteristic of new product development teams, including learning on different levels and between functions.

Questions: What are the arenas where you give feedback on each others work? How are software development problems identified, and do you improve the development method? Do you keep what works well in your development process? How are artifacts in the development process (burndown chart, backlog, daily meetings, sprint reviews and retrospectives) used in order to learn?
Scale: 0 - a practice characterized by no feedback mechanisms. 10 - continuous improvement of work methods.

3.5 Autonomy

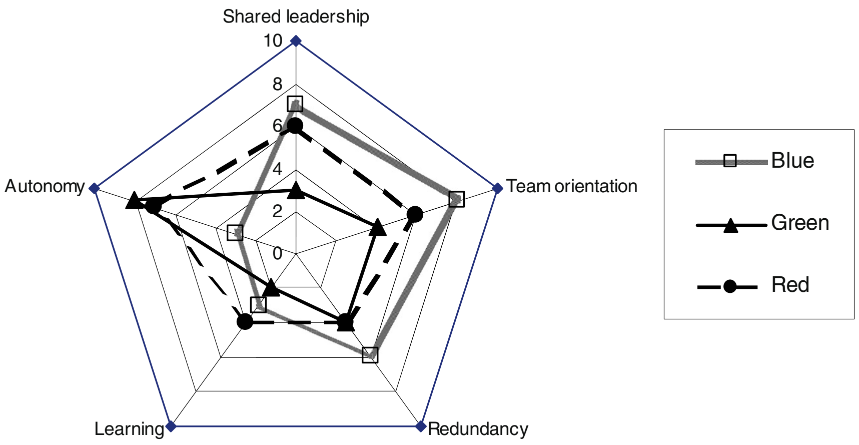
The autonomy of teams is described in socio-technical literature [26] as the team's ability to regulate their boundary conditions. Team autonomy, or external autonomy, is defined by Hoegl and Parboteeah [25] as the influence of management and other individuals (outside the team) on the team's activities. Such influence can be deliberate actions from management to limit autonomy, such as requiring the team to make certain decisions regarding work strategies or processes, project goals and resource allocation.

While some forms of team-external influence are sometimes beneficial because they provide important feedback to help project completion or encourage creativity within the team by discouraging groupthink, Hoegl and Parboteeah [25] argue that the specific type of team-external influence considered here is detrimental to teamwork in projects e.g. developing software.

Questions: Does the team loose resources to other projects? Do people and groups outside the team have influence over important operational decisions in the project? Are decisions made by the team respected by people and groups outside the team? *Scale:* 0- Decisions made by the team is not respected (by managers outside the team). 10- Decisions made by the team is respected (by managers outside the team).

4 Empirical Testing

The preliminary instrument has been evaluated in two steps. It has been presented to an agile expert group (industry and research) of 35 people. The expert group identified typical problems found in agile teams, and then applied the instrument to understand these problems. Examples are: 1) the team agrees on test-driven development, but this is not done in practice 2) the team has no control over the members assigned to the team 3) management dos not trust the team fully. The expert group found the model useful for understanding the cause and effect of such problems.



Step two was testing the instrument on three teams in two companies. The two companies were part of a larger action research program, where several companies have introduced elements from agile development. For the two companies where we tried the team radar, Scrum was introduced because they wanted to improve their ability to deliver iteratively and on time, increase the quality, improve the team-feeling and team communication. One of the companies is situated in the south of Norway and the other is situated in the northern part, therefore we name the companies “South” and “North”. Both companies had used Scrum for 18 months.

The authors conducted 17 interviews, based on the questions in chapter 3, with all the team-members in the studied projects. From this we generated radar plots and presented them to the team in a feedback session. These feedback sessions were used to discuss the score and identify improvements actions in the team, and to assess the usefulness of the instrument. All the teams confirmed the usefulness of the instrument since it brought into focus areas that needed to be improved. We will now present the main results from the interviews and feedback session from what we have called the Blue (South company), Red and Green team (North company).

4.1 South Company

This company has approximately 150 employees. They sell mass-market software and they also write customer specific software on a contract-basis. The project (Blue project) under study consists of four developers in the core team (one is a Scrum master), two GUI designers, one product owner, and one project manager. The sprints usually lasted three weeks. The team organized a 15 minutes stand-up every morning.

In the feedback session the team focused on autonomy (score 3) and learning (score 3). The main reason for a low score on "learning" was because the team often skipped retrospective and review-meetings. They felt it was challenging to do the review meeting because part of the software did not contain any GUI. Also, there was a feeling that the retrospective meetings did not really give value. One said: "the retrospective turned out to be just another nice meeting without really discussing the problems". The team decided to re-implement review and retrospectives.

The product owner constantly visited existing and potential new customers. When he got back he presented several new ideas, even during the sprint. The Scrum-master did not manage to protect the team, and this decreased the level of autonomy. One developer said: "I think the problem is that he is talking to so many customers, and then his perspective constantly changes". Changing the perspective in the middle of the sprint undermined the decisions made in the beginning of the sprint, and resulted in difficulties with delivering according to the sprint-plan. The team had problems confronting the product owner with these problems, which made it difficult to improve the processes. During the feedback session the team realised someone had to confront the product owner, if not developers could end up leaving the team. The team perceived the team-radar as useful, since it pointed out the most critical areas to improve.

4.2 North Company

This company has worked with large development projects as both a prime contractor and a subcontractor. The company possesses a stable and highly skilled staff. The company has approximately 60 employees, and the Scrum teams use 2 week sprints. Both the green and red team held short, daily stand-up meetings.

A key feature of the seven person Green team is that the members work with several different things, serving many products and projects. To some extent the team is comprised of one and two-person units - units that may have substantial overlapping interests and problems. They emphasize that being a group of some size is fruitful, to have close colleagues with which to interact regularly. As a consequence of the one or two-person specialist "mini-teams", they got a low score on redundancy (score 4), but still mutual help and overlapping competencies within the team existed. One said: "Which person the task should be assigned to is self-evident in 90 percent of the cases – it's given who does what. However no one is allergic towards doing others tasks". Another said that: "We help each other out on general tasks [not specialist tasks], so that things move faster". Related to the low score on learning (score 2), one said: "We don't do retrospectives”. Another outlined the status: “We have no systematic feedback on each others work”. This was the case because, as one member stated: “When finished with a task, then ok, you look ahead, it is more focus on the next task”. However, as one noted, “it is fun to look at others work”. In the feedback session the team found the instrument useful both for initiating important collective discussions and for highlighting potential areas for teamwork improvement. The team wanted to implement measures to address concerns related to redundancy, learning, and the relationship with the product owner and project leaders, as well as other issues.

The six person Red team was characterized by relatively high scores on shared leadership (6) and team orientation (6). On leadership, the team expressed that everyone in the team was involved when making decisions, but one expressed that ”the ones who have to do the work are most involved when making decisions”. The sprint the team was working on when we did the interviews was characterized by clear goals. However, this was due to a delivery to an external customer, and usually when having an internal product owner, the goal would be less clear, and the work would seem more dispersed. On team orientation the Red team characterized their team as having ”lively discussions”, and that ”everything is discussed”. Further, team members were said to be interested in the work of others. However some expressed that everyone did not have ownership to the team’s plans. The diagnosis of team led to a discussion with the team on topics including short and long-term planning, how they assigned work tasks, and how they viewed the role of specialists and generalists on the team. The interviews and discussion was perceived as a valuable exercise by the team.

5 Conclusion and Further Work

Within the framework of a large action research project comprising several companies using agile development methods, we have developed and used in practice an instrument for assessing agile teamwork. We found that teamwork was a major challenge in the companies [21, 29], and for improving teamwork we needed a way to describe and diagnose the status of teams and a means of tracing and engaging with change processes. Based upon the literature review and experiences from our own action research introduction and test of the agile teamwork instrument, we conclude first that both research and industry partners recognize the chosen five dimensions of the instrument as playing a pivotal role in agile teamwork. Furthermore, in addition to being an instrument suitable for diagnosing the status of teamwork in agile development, we believe it is appropriate also for tracing changes over time in the quality of the teamwork, and finally that it enables interventionist measures for improving the work in the team.

For the partners in the research project, the instrument enabled both a knowledge base and the development of a language for engaging with teamwork change processes and follow-up. From the research side, the instrument provides valuable empirical data from which to analyse and describe the status of agile teamwork, and it equips researchers with a methodological tool from which both analytical descriptions and action research interventions might ensue. The instrument also contributed in giving the researchers and developers a common language for discussing teamwork.

A possible limitation of the instrument is the process of giving a score of the teams along the five dimensions, because it might entail an idea that the score represent an "objective mapping of the world". The fine granularity of the scale (from 0 to 10) might add to this notion. Both the use of the instrument as tracing changes in the quality of teamwork over time, the scoring procedure and the scale, and the interventionist potential of the instrument will be more extensively pursued in further work. We will also investigate how the type of team and how long the team has worked together affects the scores on the instrument.

Acknowledgment

We appreciate the input received from the investigated companies. This research is supported by the Research Council of Norway under Grant 174390/I40.

References

1. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology 50, 833–859 (2008)
2. Nerur, S., Balijepally, V.: Theoretical reflections on agile development methodologies - The traditional goal of optimization and control is making way for learning and innovation. Communications of the ACM 50, 79–83 (2007)
3. Boehm, B.W., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley, Reading (2003)
4. Kraut, R.E., Streeter, L.A.: Coordination in software development. Communications of the ACM 38, 69–81 (1995)
5. Hoegl, M., Gemuenden, H.G.: Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. Organization science 12, 435–449 (2001)
6. Greenwood, D.J., Levin, M.: Introduction to action research: social research for social change. Sage Publications, Thousand Oaks (2007)
7. Guzzo, R.A., Dickson, M.W.: Teams in organizations: Recent research on performance and effectiveness. Annual Review of Psychology 47, 307–338 (1996)
8. Cohen, S.G., Bailey, D.E.: What makes teams work: Group effectiveness research from the shop floor to the executive suite. Journal of Management 23, 239–290 (1997)
9. Sapsed, J., Bessant, J., Partington, D., Tranfield, D., Young, M.: Teamworking and knowledge management: a review of converging themes. International Journal of Management Reviews 4, 71–85 (2002)
10. Katzenbach, J.R., Smith, D.K.: The Discipline of Teams. Harvard Business Review 71, 111–120 (1993)
11. Langfred, C.W.: The paradox of self-management: Individual and group autonomy in work groups. Journal of Organizational Behavior 21, 563–585 (2000)
12. Salas, E., Sims, D.E., Burke, C.S.: Is there a big five in teamwork? Small Group Research 36, 555–599 (2005)
13. Tata, J., Prasad, S.: Team Self-management, Organizational Structure, and Judgments of Team Effectiveness. Journal of Managerial Issues 16, 248–265 (2004)
14. Morgan, G.: Images of Organizations. SAGE publications, Thousand Oaks (2006)

15. Young, S.M., Edwards, H.M., McDonald, S., Thompson, J.B.: Personality Characteristics in an XP Team: A Repertory Grid Study. In: Proceedings of Human and Social Factors of Software Engineering (HSSE), St. Louis, Missouri, USA, pp. 1–7 (2005)
16. Whitworth, E., Biddle, R.: The Social Nature of Agile Teams, Agile, Washington, DC, pp. 26–36 (2007)
17. Seger, T., Hazzan, O., Bar-Nahor, R.: Agile Orientation and Psychological Needs, Self-Efficacy, and Perceived Support: A Two Job-Level Comparison, Agile, Toronto, pp. 3–14 (2008)
18. Beecham, S., Sharp, H., Baddoo, N., Hall, T., Robinson, H.: Does the XP environment meet the motivational needs of the software developer? An empirical study, Agile, Washington, DC, pp. 37–49 (2007)
19. Tessem, B., Maurer, F.: Job Satisfaction and Motivation in a Large Agile Team. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 54–61. Springer, Heidelberg (2007)
20. Acuña, S.T., Gómez, M., Juristo, N.: Towards understanding the relationship between team climate and software quality—a quasi-experimental study. Empirical software engineering 13, 401–434 (2008)
21. Moe, N.B., Dingsoyr, T.: Scrum and team effectiveness: Theory and practice. In: 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, Limerick, Ireland, pp. 11–20 (2008)
22. Kirkman, B.L., Rosen, B.: Beyond self-management: Antecedents and consequences of team empowerment. Academy of Management Journal 42, 58–74 (1999)
23. Pearce, C.L.: The future of leadership: Combining vertical and shared leadership to transform knowledge work. Academy of Management Executive 18, 47–57 (2004)
24. Hewitt, B., Walz, D.: Using Shared Leadership to Foster Knowledge Sharing in Information Systems Development Projects. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, 2005. HICSS 2005, p. 256a (2005)
25. Hoegl, M., Parboteeah, K.P.: Autonomy and teamwork in innovative projects. Human Resource Management 45, 67–79 (2006)
26. Emery, F., Thorsrud, E.: Democracy at work: the report of the Norwegian industrial democracy program. Martinus Nijhoff Social Sciences Division, Leiden (1976)
27. Marks, M.A.: A temporally based framework and taxonomy of team processes. The Academy of Management review 26, 356 (2001)
28. Takeuchi, H., Nonaka, I.: The New New Product Development Game. Harvard Business Review 64, 137–146 (1986)
29. Moe, N.B., Aurum, A.: Understanding Decision-Making in Agile Software Development: a Case-study. In: EuroMicro, Parma, Italy. IEEE, Los Alamitos (2008)

Agile Software Development and CMMI: What We Do Not Know about Dancing with Elephants

Célio Santana[1], Cristine Gusmão[1], Liana Soares[1], Caryna Pinheiro[2], Teresa Maciel[3], Alexandre Vasconcelos[4], and Ana Rouiller[3]

[1] University of Pernambuco, Departament of Computer and Systems, Benfica St. 455, 50720-001 Recife, Pernambuco, Brazil
[2] University of Calgary, Departament of Computer Science, Calgary, Canada
[3] Federal Rural University of Pernambuco, Informatic and Statistic Departament, Dois Irmãos avenue. S/N, 52171-900 Recife, Pernambuco, Brazil
[4] Federal University of Pernambuco, Informatic Centre, Prof Luiz Freire Avenue. S/N,50740-540 Recife, Pernambuco, Brazil

Celio.santana@gmail.com, capinhei@ucalgary.ca, cmgg@dsc.upe.br, teresa@cesar.org.br, lsos@dsc.upe.br, amlv@cin.ufpe.br, anarouiller@gmail.com

Abstract. In this paper we discuss how the merging of Agile Methodologies and Software Quality Models in same process today is ignoring many important aspects of both approaches. The inconsideration of these points results in a rigid integration of Agile and Quality Models that limits the full potential of their synergies. Ignoring such important items however does not necessarily means that they are not being utilized in the process, it normally indicates their utilization in an ad-hoc way. To explore this topic, we collected qualitative and quantitative data from literature and two Brazilian companies which work with agile and XP.

Keywords: Capability Maturity Model Integration, CMMI, Agile Software Development.

1 Introduction

In 2001 the Agile movement proposed a shift in the values of software development process from the mechanistic (i.e., driven by process and rules of science) to the organic (i.e., driven by softer issues of people and their interactions) [1].

Despite the changes in traditional software engineering presented by the Agile methods, in 2001 one of the CMM developers, Mark Paulk, stated that CMM and an Agile method called Extreme Programming (XP) [2] contain ideas that can be synergistic [3].

This paper provides, through the analysis of the quantitative and qualitative data gathered in literature and industry, a further contribution by bringing to attention the relevant disregarded elements when merging Agile and CMMI, as well as presents the consequences of implementing this incomplete merged methodology.

After this introductory section, section 2 explores Agile Methods. In section 3, CMMI is explained; section 4 shows how CMMI and Agile are being merged today; section 5 shows what is fully or partially unknown about this merging and section 6 shows a summary and future works.

2 Agile Methodologies

In February 2001, seventeen of the leading developers and proponents of the "Agile methods" proposed a meeting which resulted in four levels of agreement among the participants [4]:

1st Level: There was a need for methods designed to respond to change during software projects. Thus, they adopted the term "Agile" to identify such methods. They agreed that the term "Light" was not appropriate because certain projects would not employ a "Light" methodology but could still require agility.

2nd Level: The four statements of the "Agile Manifesto" [5]. These four statements capture the core values on which all of the Agile Methods are built, as well as the spirit in which they should be implemented.

3rd Level: The next level of agreement was on a set of 12 Agile Principles. In these statements, the values are fleshed out in more detail and given more concrete meaning.

4th Level: The final level of agreement, which was to be a more detailed with actual activities or tactics to running projects, was beyond their grasp at the time. They left that fourth level for each agile method to define in its own way. Some of these agile techniques are Extreme Programming (XP) [2] and Scrum [6].

3 Capability Maturity Model Integration

The Capability Mature Model Integration (CMMI) [7] is defined as a process improvement maturity model for the development of products and services. Consisting of 22 process areas where each one area can be summarized to illustrate their components, as shown in Figure 1.

CMMI enables one to approach process improvement and appraisals using two different representations: continuous and staged. The continuous representation enables an organization to select a process area and improve processes related to it using six (0 to 5) capability levels to characterize improvement related to an individual process area. The staged representation uses predefined sets of process areas to define an improvement path for an organization characterized by five (1 to 5) maturity levels.

The following rules summarize equivalent staging:

- To achieve maturity level 2, in seven process areas assigned to maturity level 2 must achieve capability level 2 or higher.
- To achieve maturity level 3 or higher, all process areas assigned to that specific maturity levels and lowers must achieve capability level 3 or higher.

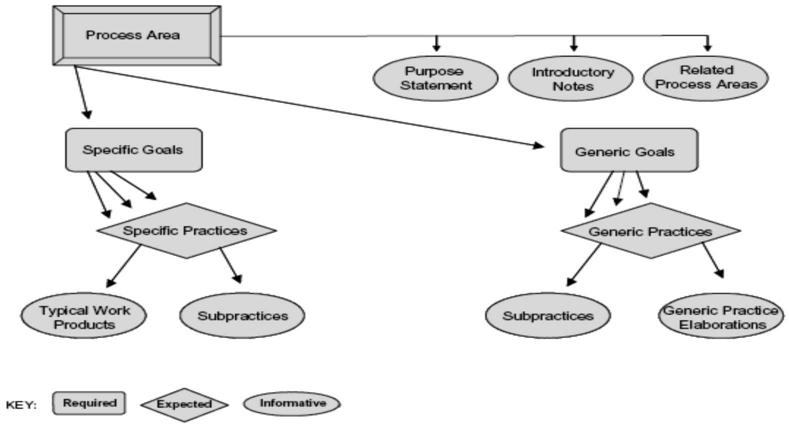


Fig. 1. CMMI Process Areas and their Components [7]

And a short description of each capability level follows:

- A capability level 0 process is defined as an “incomplete process”. One or more of the specific goals of the process area are not satisfied, and no generic goals exist for this level.
- A capability level 1 process is defined as a “performed process.” A performed process is a process that satisfies the specific goals of the process area. Although capability level 1 does not address all generic goals at capability 2.
- A capability level 2 or higher process is a performed (capability level 1) process that address all the generic goals for that capability level.

4 Agile and CMMI: What Do We Know?

4.1 Mapping Techniques of Agile Methods to CMMI Specific Practices

The first mapping between Agile and CMM was conducted by Paulk [4] fitting XP to SW-CMM level 2. Since then, many reports following Paulk’s ideas such as [8, 9] were published. These mappings refer to the low levels (2 and 3) of CMMI.

This paper also presents the research conducted in two companies using Agile and following CMMI patterns. Their names must remain anonymous due to confidentiality reasons. Company 1, located in Brazil, has eleven years of experience in the market and has recently been evaluated for SCAMPI to CMMI level 3 and uses Scrum in sixteen projects. Company 2, located in Brazil, is a small-sized company with nine years of experience in the market which will be evaluated at CMMI level 2 using XP in a pilot project. They will be referred in the following sections to provide an industry insight on the subjects.

About this subject, both companies conducted a gap analyses to understand the misfits between CMMI and Agile. These gap analyses are also the mapping of each company about agile and its relationship with CMMI specifics practices.

4.2 Traditional and Agile Software Process Improvement

For Salo & Abrahamsson [10], stated two central differences between traditional and agile SPI. Firstly, the origin of SPI goals is traditionally from the organizational level. However, in the agile project context the power for SPI lies within project teams, and their experiences throughout projects. Secondly, the process knowledge of project teams has traditionally been harvested from finished projects. Instead, the focus of agile project learning is on improving the performance of the ongoing project.

The traditional and agile SPI approaches are not contradictory. Rather, both the approaches can be beneficial in integrating agile software development and organizational learning. This, however, requires the integration of agile SPI mechanisms in the organizational learning cycle.

The companies under study (company 1 and 2) structured their Software Engineering Process Group (SEPG) outside agile teams, following the traditional SPI approaches.

5 Agile and CMMI: What We Do Not Know?

5.1 CMMI Support for Adopting Agile

According [11, 12], companies using CMMI provide support to agile making it an easier and smoother methodology to adopt than in non CMMI companies. Nevertheless, [9] stated that even presenting similar ideas, merging Agile and CMMI could be difficult, since there are other aspects seen in a different perspective.

This subject looks contradictory because the support provided by CMMI to any methodology comes from its generic practices (see Fig 1). Sutherland et. al. [13] stated how generics practices at capability 2 and 3 could support and improve Agile.

The companies under study (company 1 and 2) ignored how Generics Practices support agile following the industrial tendency of ignore this subject.

5.2 Merging CMMI Specifics Practices and Agile, Is Still Agile at all.

Mnkandla & Dowlatzky [14] define the value of agility in “allowing the concepts defined above to mutate within the parameters set by the agile values and principles”. As a matter of fact, there is no mapping that provides a full fit between Agile and CMMI. Thus, it is necessary to complement Agile with other practices, in which there is no guarantee to be following agile values and principles.

Quantitatively, [13] and [15] presented evolution of indicators of quality and productivity with the adoption of Agile merged with CMMI.

Sutherland *et. al.* [13] stated:

1. Scrum reduced defects by almost 50% compared to our previous CMMI level 5 implementation.
2. Projects using Scrum shows a 201% increase in productivity.
3. Price was reduced by 50%.when reevaluate the requirement specification.

Gravis & Jarvistock [15] Stated:

1. Defects reduced by 66%, Critical defects reduced by 79%;
2. Duration (in days) reduced by 44% and Effort (in hours) reduced by 47%;
3. Quality of Life improved by 81% business and 77% technology.

The companies 1 & 2 are still measuring their actual process (Agile + CMMI) to compare against their previous CMMI process.

5.3 Aspects Almost Totally Unknown

- *Adoption of statements and values of Agile Manifesto.*

There is no data in the literature addressing the adoption of values and statements of Agile Manifesto. Neither companies 1 or 2 considered the Manifesto on their processes.

- *Higher maturity levels of CMMI are close or deviate of Agile*

The mappings seen in section 4.1 show to the deviation between agile and process areas and specific practices of CMMI when the maturity level is raised to level 3. However, Sutherland *et. al.* [13] stated that generics practices of level 3 could improve agile. The lack of precise data on how much closer/far are Agile, specifics practices and generics practices makes it impossible to reach a conclusion about this subject.

6 Conclusion

Merging Agile e CMMI is neither strange nor innovative, many reports from both academia and industry show how Agile and CMMI could live harmoniously in the same environment with few quantitative results indicating positive results when these approaches work together.

Although, the union between lower levels of Agile Manifesto (agile methods) and CMMI (specific practices), does not provide many options for companies which want to use Agile and CMMI. Ignoring the higher levels of Agile Manifesto and the others components of CMMI, the companies could not see other opportunities to improve their process, strengthen the union of the approaches and solve commons problems.

Finally there is a few diversity of researches dealing with merging Agile and CMMI. Most of them are mappings where specific practices of the following process areas of CMMI level: Project Planning (PP), Project Monitoring and Control (PMC) and Requirements Management (REQM) are merged with Scrum or XP. This factor leads to a false impression of full understanding about the subject.

6.1 Future Works

- Make further research on how CMMI Generic Practices could support and improve Agile, and establish a “distance” between them;

- Observe the behavior of these distance when higher maturity levels of CMMI are achieved ;
- Collect indicators such as quality, productivity, schedule and costs after the merging and verify their evolution comparing with themselves before the merging;
- Investigate how the Agile Manifesto could be considered in this union.

References

1. Mnkandla, E., Dwolatzky, B.: Balancing the human and the engineering factors in software development. In: IEEE AFRICON 2004 Conference, pp. 1207–1210 (2004)
2. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2000)
3. Paulk, M.: Xp from a CMM perspective. IEEE Software 18(6), 19–26 (2001)
4. Koch, A.S.: Agile Software Development - Evaluating the Methods for Your Organization. Artech House, Boston (2005)
5. Manifesto, A.: Agile Manifesto for Software Development (2001), `http://www.agilemanifesto.org/`
6. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Englewood Cliffs (2000)
7. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI®: Guidelines for Process Integration and Product Improvement. Addison-Wesley, Reading (2003)
8. Nawrocky, J., Walter, B., Wojciechoeski, A.: Comparison of CMM Level 2 and eXtreme Programming. In: Kontio, J., Conradi, R. (eds.) ECSQ 2002. LNCS, vol. 2349, p. 288. Springer, Heidelberg (2002)
9. Käkhönen, T., Abrahamssom, P.: Achieving CMMI Level 2 with Enhanced Extreme Programming Approach. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 378–392. Springer, Heidelberg (2004)
10. Salo, O., Abrahamssom, P.: Integrating agile software development and software process improvement: a longitudinal case study. In: International Symposium on Empirical Software Engineering (2005)
11. Turner, R., Jain, A.: Agile Meets CMMI: Culture Clash or Common Cause. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 153–165. Springer, Heidelberg (2002)
12. Jakobsen, C., Johnson, K.: Mature Agile with a twist of CMMI. In: Proceedings of the Agile Development Conference, pp. 212–217 (2008)
13. Sutherland, J., Jakobsen, C., Johnson, K.: Scrum and CMMI Level 5: The Magic Potion for Code Warriors. In: Proceedings of the Agile Development Conference, pp. 466–471 (2007)
14. Mnkandla, E., Dwolatzky, B.: Balancing the human and the engineering factors in software development. In: IEEE AFRICON 2004 Conference, pp. 1207–1210 (2004)
15. Jarvis, B., Gristock, S.: Extreme Programming, Six Sigma & CMMI – How they can work together, a JP Morgan Chase Study, `http://www.sei.cmu.edu/cmmi/presentations/sepg05.presentations/jarvis-gristock.pdf`

Is ISO/IEC 15504 Applicable to Agile Methods?

Giuseppe Lami[1] and Fabio Falcini[2]

[1] Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione
via Moruzzi, 1 – I-56124 Pisa, Italy
giuseppe.lami@isti.cnr.it
[2] Intecs SpA
via E. Giannessi, 5 – I-56121 Pisa, Italy
fabio.falcini@intecs.it

Abstract. In the last two decades several models for evaluating software process capability have been defined and became more and more popular. The application of such models, and in particular the ISO/IEC 15504, determined a general software process improvement in many domains. Nevertheless, the application of the ISO/IEC 15504 standard is still considered by many agile developers as incompatible with agile approaches. Such an attitude is mainly based on common misunderstandings on what the ISO/IEC 15504 is and on what its application involves. This paper aims at showing that this standard, if genuinely applied, can be effectively used also in agile contexts.

Keywords: Software Process Capability Determnation, ISO/IEC 15504.

1 Introduction

After over 30 years of software engineering, software development is today supported by disciplined and formal methodologies, paradigms and tools. Models and standards providing guidelines, processes and evaluation methods for software development exist as well. In particular, in the past two decades, several models for improving software development processes became popular (in particular CMMI and SPICE – ISO/IEC 15504). We witnessed, in our experience, some misuses of such models that determined, sometimes, the paradox to be perceived as a bundle to the development activities in software development teams. Agile methods arose as a reaction to the misuse of firm and disciplined approaches to the software development.

In such a situation two parties rose up in the software community: the "agilers", challenging the disciplined approach seen as a way to spend the effort most on the documentation than real software (the product), and the "orthodox" believing that the major effort required by a disciplined, rigorous and documented approach pays in terms of quality of the final product. The authors of this paper (being representatives of both the two parties of the barricade) discussed, on the basis of their common experience in leading software process improvement and assessment initiatives, how and if the disciplined and rigorous approach and the agile one can co-exist. In this paper, the applicability of the ISO/IEC 15504 (also known as SPICE) [1] for software process assessment and improvement to agile contexts is discussed. The SPICE model is generally perceived in the software community as a way to evaluate software process from a formal and document-based perspective. Our aim is to debunk such a myth and to show that the SPICE model, if genuinely applied, can be effectively used also in agile organizations.

This paper is structured as follows: in Section 2 we present the ISO/IEC 15504 standard. In section 3 some typical misunderstandings about what ISO/IEC 15504 is and what it requires to be applied are discussed. In Section 4 the applicability of the ISO/IEC 15504 standard to agile contexts is systematically analyzed. Finally, in Section 5 the final discussion is provided as well as the conclusions.

2 ISO/IEC 15504: Key Concepts

It is not the aim of this section to provide a detailed description of the ISO/IEC 15504 standard [1]; what we are interested in is to let the reader able to understand the basic concepts underlying the standard. The purpose of the standard is to provide a scheme for evaluating the capability of the software process and a way to improve it. *Process capability* is defined as a characterization of the ability of a process to meet current or projected business goals.

The three basilar concepts of the standard are: Process Reference Model (PRM), Process Assessment Model (PAM) and Measurement Framework.

PRM: It is a model comprising definition of processes in a lifecycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between processes. In other words, the PRM is the set of the descriptions of the processes that will be assessed. While the standard doesn't include any specific PRM (it defines the requirements for defining a PRM), in practice the ISO/IEC 12207 [2] standard is very often used as PRM for ISO/IEC 15504.

PAM: It is a model suitable for the purpose of assessing process capability, based on one or more PRMs. A PAM provides a two-dimensional view of process capability. In one dimension, it describes a set of process entities that relate to the processes defined in the specific PRM; in the other dimension the PAM describes capabilities that relate to the process capability levels and process attributes defined in the Measurement Framework that is part of the standard.

Measurement Framework: It provides a schema for use in characterizing the capability of an implemented process with respect to the PAM. Capability is defined on a six-value ordinal scale. The scale represents increasing capability of the implemented process. For the description of the meaning of each capability level, refers to [1]. The achievement of a certain capability level is established by the rating of specific Process Attributes (i.e. measurable characteristics of process capability applicable to any process). The extent a Process Attribute is achieved is measured using a four-value rating scale.

From what stated above, it should be clear that ISO/IEC 15504 is a standard having the purpose of providing a structured approach for the assessment of processes. It is neither a software development paradigm, nor a source of practices to be adopted, nor a standard to be compliant with.

3 The Perception of SPICE: Myths and Truth

The authors collected, on the basis of their experience as consultants for software process improvement initiatives and as SPICE assessors, common imprecise perceptions about SPICE arising from software developers once it is proposed for process assessment or improvement. In the following typical statements about SPICE developers say, are listed and commented in order to show that they are largely not justified and often based on a misunderstanding of the real nature of SPICE.

"SPICE requires being compliant with the V-model software development"

- False. It is true that the very most common PRM associated to the ISO/IEC 15504 is the ISO/IEC12207 that contains a set of processes that *de-facto* reflects those of the V-model for software development. Nevertheless, the sequence and the importance of the processes defined in the ISO/IEC 12207 are not determined in the standard. Moreover the association between SPICE and a PRM for software development may vary. Every organization can modify the ISO/IEC 12207 PRM or define a new one, without missing the SPICE compliance.

"SPICE requires the performance of a pre-defined set of Base Practices"

- False. The Part 5 of the ISO/IEC 15504 includes some Base Practices but they are not mandatory. In fact, ISO/IEC15504 Part 5 is an exemplar part of the standatd. What the standard requires to achieve the Capability Level 1 is the fulfillment of the Process Purpose by means of the achievement of the related Process Outcomes, no matter what practices are used to do that.

"SPICE requires the production of a set of mandatory documents"

- False. The considerations made for the point above, are valid also for documents. SPICE does not require producing any specific document; it only requires producing documents appropriate for the project characteristics and the organization's business goals.

"SPICE – capability level 2 requires the production of a formal project plan document"

- The sentence is partially true. There is no requirements on the degree of formalism of the project plan document (it depends on the specific needs of the organization and projects), but there is the request to have the processes planned in terms of activities and performance to reach the Capability level 2.

"The compliance with SPICE determines an overload of documents"

- False. SPICE does not necessarily produce any overload of documents but only the necessary documents for achieving the target capability level. The ability of the assessors/improvers is to understand, by taking into account the specific business needs, application domain and operational environment, what are the essential documents required, without impose any superfluous effort.

"SPICE – level 3 requires following the same defined process for all the projects independently of their size and complexity"

- False. Having a defined general process is required only at Capability level 3. Having a defined process doesn't mean that such a process shall be replicated for every project independently of its size and complexity. In fact, the process definition shall be accompanied by the capability to deploy processes in a tailored manner according to the specific project characteristics.

4 Agile vs. SPICE: Theory and Practice

In this section we analyze the applicability of the SPICE model to Agile organizations. Several different Agile methods exist. Each agile method has its own characteristics and peculiarities [3, 4]. We do not consider any specific method for our analysis, the applicability of SPICE in agile contexts is discussed from a generic point of view considering the agile methods' common aspects and principles.

To perform the analysis we consider the five levels the capability dimension the ISO/IEC 15504 standard PAM is composed of. For each capability level we discuss if, in principle, possible drawbacks in applying SPICE to the agile methods exist.

Level 1: As discussed in section 3, to reach the capability level 1, a process needs to achieve its stated purpose. In general, the capability level 1 can be achieved no matter what techniques, methods or tools are applied. The point is to understand if the agile approaches can be able to achieve the purpose of the processes in the PRM in the same way the traditional approaches do. Agile practices are not always new, they are traditional practices applied with a different orientation and with the emphasis given to specific aspects (as the face-to-face communication, the high frequency of releases,…) [5, 6]. In addition, there are several works in literature [7, 8, 9] showing that the agile approaches con co-exist with a typical model based on a traditional and process-based software development: CMMI [10]. Moreover, SPICE is currently and successfully applied to organizations adopting the iterative model for software development [11]. The iterative approaches, in comparison with the waterfall or V models [12], can be considered bringing a certain degree of agility [13]. Agile can also be seen as an extreme application of the iterative approach.

Level 2: To be rated at capability level 2, the activities related to specific process shall be planned, monitored and, possibly, adjusted; moreover its work products shall be appropriately established, controlled and maintained. Agile processes are performed paying more attention to answering questions than following a plan as well as to short term planning. Such a principle contained in the Agile Manifesto [14] seems to be in contrast with the requirements of SPICE for level 2. On the contrary, SPICE at capability level 2 focuses on the capability to adapt the plans to changing situations and to establish the right resources allocation to the project; these aspects are fitting well with agile approaches. Moreover, SPICE requires paying attention to the quality requirements and management of Work Products. In the ISO/IEC 15504 standard the term Work Product is used in place of document. The reason is the will to refer to general artifacts produced during the software development (not only documents) and the will to emphasize the working nature of the products. An essential requirement to be satisfied at capability level 2 is the appropriate production, usage and control of work products (i.e. in a way being in line with the project characteristics and needs). Again, such an approach fits well for agile contexts.

Level 3: To be rated at level 3 a process shall be managed and implemented using a defined process that is capable of achieving the expected outcomes. Agile processes have the characteristic to be adaptable and flexible in order to be suitable for changing contexts. SPICE, even if it requires a defined process at capability level 3, requires at the same time the capability of tailoring (i.e. adapt the defined process) according to the specific project needs and context. Without such a capability the capability level 3 cannot be achieved. The agile processes can achieve the SPICE capability level 3 without any particular limitation.

Level 4: To achieve capability level 4, a process is required to use measurements to ensure that the performance of the process supports the achievement of process objectives in support of defined business goals. Agile processes are, for example, oriented on using the number of customer requirements implemented as the main measure of the progress and achievement of the objectives. SPICE doesn't indicate what the measures to be collected and used are, and then there is not incompatibility also in this case. Moreover, to achieve this capability level it is required to use the measures collected for controlling and possibly adjusting the process performance. That, again, is not in contrast with the agile philosophy.

Level 5: To achieve capability level 5, the process shall be continuously improved to meet the current and projected business goal. That is a target for any organization adopting any development methodology, then also for agile organizations.

We showed that ISO/IEC 15504 is, in principle, applicable to agile contexts, nevertheless it doesn't mean that it can be in practice applicable without encoutering in practical problems. Below, we identify and discuss some problems that should be solved to make SPICE assessments practically executable in agile organizations.

Process Reference Model: we already noted that the SPICE's most used PRM is the ISO/IEC12207. We also discussed in the previous sections that it can, in principle, be applied to agile processes. Nevertheless, the ISO/IEC 12207 may be not suitable for specific contexts. In the past, several initiatives have been undertaken with the aim of personalizing SPICE in order to make it able to fit well in specific application domains. In particular, SPICE has been tailored for the space, automotive, banking and medical domains. All these initiatives arose from the difficulties to apply generic SPICE PRMs to specific domain having some peculiarities that required a tailored model. The most successful initiative (i.e. the today's most widely applied in the specific domain) is the Automotive SPICE [15, 16]. The experiences in defining specific PRMs indicate the way for defining an Agile-specific PRM in order to apply the SPICE assessment mechanism to a set of processes specifically defined and then well focused on the agile methodologies.

Assessor's competence: An agile-specific assessors' qualification scheme should be build-up. The assessors should be trained appropriately in order to let them ready to consider the nature of agile approaches and consequently be able to rate the processes objectively and in repeatable way. The assessors of agile processes should be aware of the peculiarities of the agile methodologies and have direct experience of software development in agile contexts.

5 Final Discussion and Conclusions

The experience of the last decade shows that the availability of effective models for quantitatively evaluating the capability of software organizations is very important for many reasons. First because it allows to determine the risk of having low-quality software products *a priori* (i.e. before starting a project) on the basis of the evaluation of the software development process. Moreover, it allows the setting up of process improvement programs by identifying targets in terms of process capability. These advantages can be moved also in the *agile world*. We discussed the applicability of SPICE in agile contexts starting from the objection of some common misunderstanding, and false myths on the real nature of the SPICE approach. Then, in a more systematic way, we highlighted the suitability of the SPICE model for agile contexts as well as the barriers to be overcome in order to let it be practically applied. By choosing an agile approach, an organization should not be prevented to assess its "maturity" using the SPICE model, and to take corresponding improvement steps. The discipline of higher maturity levels over agile projects would combine the best of discipline and agility with an optimal trade-off in many contexts between cost, quality and time-to-market. In conclusion, we are optimistic about the applicability of the SPICE model to agile contexts, and we consider this paper as a contribution to get the *agilers* and the *orthodox* closer in order to mutually take advantages.

Acknowledgements

Thanks to John Favaro and Paolo Panaroni of Intecs Spa for the precious suggestions and support.

References

1. International Organization for Standardization. ISO/IEC 15504 International Standard Information Technology – Software Process Assessment: Part 1–Part 7 (2008)
2. International Organization for Standardization. ISO/IEC 12207 International Standard System and Software Engineering – Software Life Cycle Processes (2008)
3. Bohem, B., Turner, R.: Balancing Agility with Discipline – A Guide for the Perplexed. Addison Wesley, Boston (2004)
4. Cohen, D., Lindvall, M., Costa, P.: Agile Software Development. DACS SOAR Report n. 11. Data & Analysis Center for Software, Rome, NY (2003)
5. Cockburn, A., Highsmith, J.: Agile Software Development: The People Factor. IEEE Computer 34(11), 131–133 (2001)
6. Highsmith, J., Cockburn, A.: Agile Software Development: The Business of Innovation. IEEE Computer 34(9), 120–122 (2001)
7. Glazer, H., et al.: CMMI or Agile: Why Not Embrace Both! Technical Note CMU/SEI-2008-TN-003. Carnegie Mellon University - Software Engineering Institute (2008)
8. Paulk, M.C.: Agile Methodologies and Process Discipline. CrossTalk: the Journal of defense Software Engineering 15(20), 15–18 (2002)
9. Paulk, M.C.: Extreme Programming from a CMM Perspective. IEEE Software 18(6), 19–26 (2001)
10. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI Guidelines for Process Integration and Product Improvement. Addison-Wesley, Reading (2004)
11. Larman, C., Basili, V.: Iterative and Incremental Development: A Brief History. IEEE Computer 36(6), 47–56 (2003)
12. Sommerville, I.: Software Engineering, 6th edn. Addison-Wesley, Reading (2001)
13. Larman, C.: Agile and Iterative Development: A Manager's Guide. Addison Wesley Professional, Reading (2003)
14. Beck, K., et al.: The Agile Manifesto (2001), http://www.agilemanifesto.org
15. Automotive SIG: Automotive SPICE Process Assessment Model (PAM), rel. V.2.4 (2008)
16. Automotive SIG: Automotive SPICE Process Reference Model (PRM), rel. V.4.4 (2008)

Lesson Learnt from an Agile Implementation Project

Paul Murphy and Brian Donnellan

National University of Ireland, Galway, Ireland
paulmurphyster@gmail.com

Abstract. This paper focuses on the communication concerns that unfolded in a company as it endeavored to move from its existing waterfall model to an agile environment. A task team was established to adopt an agile practice and to trial it on a customer specific project. The team was given the freedom to be innovative and adopt whatever agile practices they wished. Early enthusiasm was evident however management had concerns about viability of this new approach. Management did not see the project plans and corresponding documentation trail of the waterfall model, and from their perspective it lacked structure. Trust between management and engineers weakened primarily due to a lack of communication and a reduction in the ability to work together, resulting in the waterfall model being reinstated.

Keywords: Communication, Collaboration.

1 Introduction

This paper focuses on the good and bad aspects of communication that evolved when a company attempted a transition from a waterfall model to an agile environment. A very brief synopsis of the literature review and research methods is presented due to the restrictions on the permitted size of this paper. The main focus will be on the findings, followed by a discussion and a conclusion

2 Literature Review

As this paper focuses on communication it will consider two of the four perspectives as outlined by the Agile Alliance which was established in 2001. This is in relation to "individuals and communication over processes and tools" and "working software over comprehensive documentation". Many authors have seen communication as important for innovation [7], [12], [19]. The Agile Alliance does not see wide-ranging documentation as important as working software however the dissemination of knowledge through documentation or a knowledge management system is also a requirement of innovation [2], [6], [7], [9], [10], [20]. Communication between everybody is vital for an organisation to be innovative so that everybody understands what the other is doing [2].

Agile Methodologies and Practices. Each agile methodology attempts to accommodate various factors that the authors perceived to be important in order to develop an agile approach. Such characteristics include the amount of interaction a customer will have with development, the size of the team and the size or type of system being developed [1], [4], [8]. Other factors include whether a team is co-located or distributed. As can be seen later in the findings all of these have an effect on communication. XP was embraced by the agile team. It is not within the scope of this paper to go into details on the agile practices used however details can be found in the literature [4], [14], [17], [18].

3 Research Methods

The research was on a multi-national company that was in existence for over 25 years and during this time had developed a well documented waterfall model. As the company grew so too did the complexity of projects and documentation. A group of engineers were looking for a better more efficient way of working, and so proposed to adopt the XP methodology. Management provided them with a specific customer project to trial run the new way of working. A number of people were contacted that were involved in this project which included engineers, architects, project manager and product managers. In order to provide a comparison on responses a questionnaire was circulated [15]. The questionnaire helped to identify the type of person (e.g. developer or architect) to see if there were any trends. There were also a number of open ended questions. If further clarifications were required this was conducted via a Skype call, MSN or email. Comments from individuals have been included in quotations particularly when others had similar thoughts. While a cross comparison on all responses was undertaken in relation to agile methods and practices, the working environment, training, and project planning the focus of this paper will be on the communication between peers and management.

4 Findings

A number of aspects were considered when analysing the data and are presented in each sub-chapter below however it must be remembered that each view can influence the other. The chapter concludes with a summary from participants in relation to their views on the future of agile

Initial Take Up Of Agile: The agile initiative in the company was undertaken by 4 engineers who were tired of the "bureaucratic overhead of a heavy process nobody actually followed" in reference to the waterfall model and all its documentation and processes which they referred to as "analysis paralysis". One director supported them in this initiative.

The Agile Team: The team started with 4 people with a development background and grew to include architects, developers and project/product managers to about 30 people. When test, documentation and sales were added the team grew to 55 people. Agile contractors were also recruited to speed up the development. Universities and Higher Education Institutions were not contacted to help support or develop the agile team. The large team was dispersed across company subsidiaries and it was reported that "intra-team communication became difficult, with lots of communication issues due to multiple sites within multiple countries".

Training: While the majority of the team received training it was reported that the product managers did not and they stuck to the old waterfall method of trying to clarify all requirements up-front.

The Environment: The team was given freedom to establish their own work environment in a corner of the building which they setup as one room with glass partitions so people from the outside could look in. On a positive note this led to increased communication between those in the office. From a negative view it was reported that the environment was very noisy and no room left for people to think on their own. The overhearing of irrelevant discussion was also deemed distracting by some and anyone in a "bad mood quickly brought the whole team down".

Communication: Internal to the corner office, face to face communication was possible. Externally, information was conveyed via meetings and a wiki. Internally the team believed it was easy to come and discuss issues while external team members reported that they felt "uncomfortable going to the corner office". Some people considered that this gap in communication was one of the main reasons as to why agile development was dropped. A communication rift developed between the product mangers who were external to the corner office and the developers who were in the corner office. As stated "neither of the groups knew or wanted to learn how the other group works and what they expected, one was writing thick product specifications when the other was expecting simple user stories".

Project Planning: Management were used to the waterfall model project plans and lost confidence as they could not determine the percentage of work completed and what stage gate they were at. Management was also used to a delivery date, costs analysis and statistics on progress being presented however this was also not available. From the development and project planning perspective a deliverable product after each iteration was not available. Had one been available it would have restored management's confidence.

The Customer: Product management was the customer interface however it was reported that they could not prioritise the features to determine which should be developed first. Only when management placed a product manager inside the corner office did communication improve. On a positive perspective the customer got exactly what they wanted. When management decided that they wanted to make a generic product available they resorted to a waterfall model. The specific customer development was reported as a disadvantage for the future generalisation of the product.

Agile Methods and Practices: Not everybody believed they were following XP, some thought it was Crystal and others Feature Driven Development. Similarly, not everybody followed the same practices. The more popular practices included user stories, stand-up meetings, pair programming and continuous integration of code. Only half of the project team knew about team coding standards. Very few of the team knew the big picture yet the system metaphor suggest that it is "one of the most important practices of all" [14], and as reported each sub-team "focused on their own limited problem area".

Agile Today: Due to management's "lack of control" of the project the agile development was stopped. Management liked some aspects of agile e.g. stand up meetings. For New Product Development (NPD) agile would be a good option however for multiple customers and versions they would feel more comfortable with the waterfall model.

5 Discussion

The following chapter reviews the findings on communication from an innovation perspective through an organisation and environment lens.

Organisation Innovation: The team was supported and empowered by top management [11], [20] and provided with direct communication to a director in the company. "Empowerment, when combined with leadership support and commitment, gives people freedom to take responsibility for innovation" [11]. While some complained about overhearing conversation, such close collaboration should be embraced [18]. They initially started with 4 people however this expanded to 30 and up to 55. This trumpet model of development [2] meant that the whole team could not be co-located resulting in a globally distributed agile development environment which made XP unsuitable [1], [13]. A lack of understanding of the agile principles and practices between management and engineers led to poor communication in three directions, top-down, bottom-up and diagonally [16]. Organising the groups into specialties, product managers and developers leads to one not understanding the others needs and concerns [2]. Instead of the wiki, face to face communication would have been far more effective [2], [3]. Senior management were use to milestones from the waterfall model and this led to concerns about deadlines which has also been found in other research and requires a change in culture [1], [5].

An Innovative Environment: Placing the initial agile team into an office promoted communication amongst them, and letting them design their own office created a team spirit however external communication appeared to suffer. If the office was located centrally without glass barriers this might increase the communication and ”chance encounters, which create the possibility for inspiration and creativity - the sources of innovation” [2].

6 Conclusions

Throughout the findings a break down in communication was evident. Isolating the agile team to a corner office with glass barriers did not help to break the barriers and encourage face to face communication. Three aspects of communication have been seen as important for innovation, ”communication for coordination, communication for information and communication for inspiration” [2]. It appears from the case study that all three were problematic. This could have been improved if the corner-office was more centrally located and accessible.

Acknowledgments. I wish to acknowledge the participants. Thanks to Andrew Begel and Nachiappan Nagappan from Microsoft Research for providing a questionnaire, which they used in their survey and influenced the questions used in this case study.

References

1. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile software development methods: Review and Analysis. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478, pp.7–94 (2002)
2. Allen, T., Henn, G.: The Organization and Architecture of Innovation: Managing the Flow of Technology. Elsevier, Oxford (2007)
3. Anderson, N., Jarskog, J.: Front End of Innovation. A study of how mature companies can improve the initial stages of innovation, Master Thesis at Linköping Institute of Technology, Linköping University Department of Computer and Information Science (2002)
4. Beck, K.: Extreme Programming Explained. Addison-Wesley, Boston (2000)
5. Begel, A., Nagappen, N.: Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study (2007)
6. Chesbrough, H.: The Era of Open Innovation, MIT Sloan Management Review (2003)
7. Denning, P.: The Social Life Of Innovation. Communications of the ACM 47(4), 15–19 (2004)
8. Cockburn, A.: Writing effective Use Cases: The Crystal Collection for Software Professionals. Addison-Wesley Professional, Reading (2000)
9. Donnellan, B., Fitzgerald, B.: A Knowledge Management Application to support Knowledge Sharing in a Design Engineering Community. In: Proceedings of 11th European Conference on Information Systems (ECIS), Naples, Italy (2003)
10. Drucker, P.F.: Innovation and Entrepreneurship: Practice and Principles. Heinemann, London (1985)

11. Flynn, M., Dooley, L., O'Sullivan, D., Cormican, K.: Idea Management For Organisational Innovation. International Journal of Innovation Management 7(4), 417–442 (2003)
12. Kautz, K., Nielsen, P.: Understanding the implementation of software process improvement innovations in software organizations. Info. Systems Journal 14, 3–22 (2004)
13. Layman, L., Williams, L., Damian, D., Bures, H.: Essential Communication Practices for Extreme Programming in a Global Software Development Team. Information and Software Technology 48, 781–794 (2005)
14. McBreen, P.: Questioning Extreme Programming. Addison-Wesley, Reading (2003)
15. McCracken, G.: The Long Interview Newbury Park. Sage Publications, CA (1988)
16. Rollinson, D.: Organisational Behaviour And Analysis An Integrated Approach, 3rd edn. Prentice Hall, Englewood Cliffs (2005)
17. Sharp, H., Robinson, H.: An Ethnographic Study of XP Practice. Empirical Software Engineering 9, 353–375 (2004)
18. Sharp, H., Robinson, H.: A Distributed Cognition Account of Mature XP Teams. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 1–10. Springer, Heidelberg (2006)
19. Slappendel, C.: Perspectives on innovation in organizations. Organization Studies 17(1), 107–129 (1996)
20. Tidd, J., Bessant, J., Pavitt, K.: Managing Innovation: Integrating Technological, Market and Organizational Change, 3rd edn. John Wiley and Sons, Chichester (1997)

A Study of Risk Management in DSDM

Sharon Coyle and Kieran Conboy

Business Information Systems Group, Centre for Innovation and Structural Change,
J.E. Cairnes Graduate School of Business and Economics,
National Universtiy of Ireland Galway
{Sharon.Coyle,Kieran.Conboy}@nuigalway.ie
http://www.nuigalway.ie/commerce/

Abstract. A principle objective of agile methods is to reduce well-known risks associated with common systems development project failures. While there is extensive academic literature on risk management and its growing importance, literature in relation to risk management in an agile context is still in infancy. The purpose of this research paper is to highlight the extent to which risk management practices are incorporated into a specific agile method known as DSDM. The methodology deployed for this research involved a case study of a change management consultancy firm dedicated to the use of the Dynamic Systems Development Method (DSDM).

Keywords: Dynamic Systems Development Method (DSDM), Information Systems Development (ISD), Risk Management, Agile Methods.

1 Introduction

The continuing growth in the utilization of agile methods highlights a necessity within organizations to adapt and respond to change at a more efficient pace. Furthermore, the use of such methods is primarily driven by a strong desire to reduce well-known risks (such as scope creep), associated with common system development project failures. However, no matter what the nature of change, there will always be associated risks involved and it is therefore imperative that risk management procedures are in place. Agile methods are known for their use of iterative development, active user involvement and their acknowledgement of the need to incorporate changing system requirements and "focus on generating early releases of working products using mostly collaborative techniques" [1].

1.1 Motivation for Research

While there is extensive literature on risk management, research of risk management in agile ISD projects is non-existent. This is surprising considering how quickly agile methods are being adopted in ISD where a survey conducted by Vijayasarathy and Turk [3], having a total of 98 respondents indicated that sixty percent use agile approaches in seventy five percent or more of their projects. Many books on agile methods "have remarkably little to say about how a development team determines the risks it faces, prioritises them or takes action to negate their effects" [4]. Essentially agile approaches must tailor traditional risk management techniques meant for "years-long projects into a risk driven agile iteration lasting only seven to thirty days" [4]. How agile projects go about doing this remains unknown.

Numerous agile methods exist and a detailed analysis of risk management across all these methods was beyond the scope of this research. DSDM was chosen for this study as it is considered to be the first "truly agile software development method" [2]. It is also a worthy example to use because the method not only focuses on systems development from a coding or technical perspective but also places emphasis on higher-level business perspectives. Specifically the research focuses on three main elements of risk management, namely risk identification, estimation and evaluation.

2 Theoretical Foundations

While many authors highlight distinct approaches and frameworks for dealing with risk management, its basic fundamentals remain the same and are consistent across disciplines. The literature shows similar emphasis on the most important activities in risk management namely, those identified by Rowe [6] and Charette [7] – the early practitioners of risk management – who outline the three main elements of risk assessment as (i) Risk Identification, (ii) Risk Estimation and (iii) Risk Evaluation.

2.1 Risk Identification

Risk identification is the reduction of descriptive uncertainty [6] which involves "surveying the range of potential threats" [7]. This element of risk assessment involves detecting issues which could jeopardize or threaten the success of a project [8, 9]. Chapman [10] states that "the risk identification and assessment stages have the largest impact on the accuracy of any risk assessment." It is therefore the most important stage of risk management and is an early, yet ongoing, continuous process that requires regular screening and monitoring [9, 11].

The two most dominant sources of internal risk identified across the literature are Senior/Project Management and Project Team. The dominating external source of risk is the client. All of these were collectively identified by Mantel, Meredith et al. [12]. Every source of risk can have numerous risk factors. A risk factor is "a condition that forms a serious threat to the completion of an IT project" [13]. Wiegers [14] identified some of the most dominant risk factors. For example, creeping user requirements and excessive schedule pressure placed eighty and sixty five per cent of system development projects at risk respectively.

2.2 Risk Estimation

Risk estimation is the reduction of measurement uncertainty [6] where "the values of the variables describing the system are determined, the various consequences of an event occurring are identified" and finally, "the magnitude of the risk is determined" [7]. Risk estimation attempts to estimate "the chance (or probability) of potential loss" as well as "the exposure to potential loss i.e. the consequences or magnitude of the identified risks" [16]. The chance of potential loss is essentially the process of attaching a probability of occurrence to any identified risk. As Hall [5] states “estimation is the appraisal of risk probability and consequence.” Consequence is decided relative to cost, schedule and technical goals [5]. According to McManus [17], probability data should be used to compute the risk. When no actual data on probabilities exist, estimates by individuals most familiar with the project, its risk factors and overall problems are a good substitute [12].

2.3 Risk Evaluation

According to Rowe [6], risk evaluation involves “risk aversive action, which can result in risk reduction or risk acceptance.” Risk aversive action is essentially any mission that is undertaken to control a risk [6]. At a glance, the purpose of risk evaluation is to amalgamate the results of the risk estimation phase [15] and then decide the best action to take. Risk evaluation involves (i) establishing the ‘acceptable’ level of risk, (ii) understanding how the risks interact and (iii) determining the action(s) that needs to be taken [18] such as risk mitigation, risk avoidance, risk acceptance or risk transfer [15].

The acceptable level of a risk will depend on “individual propensity to take risks” [6], which lie in the hands of the project manager and as a result is difficult to specify. By avoiding any risk action however, we are eliminating any proactive risk management. This means that if the risk occurs, reactive approaches will be applied such as contingency planning [19].

2.4 Overview of DSDM

The main idea behind DSDM “is that instead of fixing the amount of functionality in a product and then adjusting time and resources to reach that functionality, it is preferred to fix time and resources and then adjust the amount of functionality accordingly” [2]. The DSDM Consortium advocates that because each organization is different none of its practices are detailed [2]. DSDM phases are made up of functional design prototype iteration followed by actual implementation with each iteration stage identifying, agreeing, creating and reviewing the prototypes [20]. However before any iteration begins, a project will always start with a feasibility and business study. This highlights that that the method is not just focused on technical deliverables and ensures that the business is the driver of technological developments.

DSDM has an active community and although the method is fully available, access to white papers outlining the specific use of the method is limited to such members [2]. Therefore, the extent to which DSDM incorporates risk management in practice is unknown. Empirical evidence of the method is enclosed in such white papers, which are not publicly available. For the purpose of this research, a member of the DSDM Consortium who participated in this study, kindly provided the researcher access to information regarding DSDM’s fundamental principles for conducting risk management.

2.5 Risk Management in DSDM

Risk management practices conducted in DSDM projects are driven by the following processes. A *suitability risk list* determines at the outset how compatible a particular project is for the use of DSDM. A *risk log* is then created, maintained and updated throughout the life of the project [21]. The DSDM Consortium [21] has stated how "systems that meet the needs of the business are delivered through the incremental and iterative approach with its continuous feedback from users" while "cost and time overruns are avoided by the use of timeboxes." Therefore timeboxing is used to reduce one of the main risk factors in systems development i.e. scope creep.

As a result many would propose that methods like DSDM would not need to incorporate a high degree of risk management into their processes because the very reason for their existence is to reduce common risks associated with these well-known project failures. However, it appears from this literature research from the DSDM Consortium that DSDM itself gives no less consideration to risk management than what has been documented for traditional projects.

3 Research Methodology

This research utilized a single case study focusing on an Irish change management consultancy firm, XpertResults (pseudonym) that have utilized DSDM since its existence over eight years ago. All participants in the study have extensive experience using traditional systems development methodologies and could therefore make accurate comparisons between the practice of risk management on both agile and traditional systems development projects.

A qualitative research approach was chosen because of its greater exploratory nature and its applicability to this research domain as it focuses "on gaining familiarity with the subject area" and gaining "insights for more rigorous investigation at a later stage" [22].

3.1 Data Collection

Due to the fact that there was little pre-existing theory on the phenomena being studied [23] the data collection technique needed to "emphasise meanings and experiences related to the phenomena" under investigation [22]. As such, for the purpose of conducting primary research the author decided to use semi-structured, personal one-to-one interviews, which were carried out with XpertResults' managing director as well as four of its key consultants (having extensive software development experience) and analysts. Each interview had a duration of approximately one hour.

4 Findings and Analysis

During data analysis it became evident that the concept of managing risk can never be curtailed whether the project is agile or traditional based. In fact, one of the consultants interviewed emphasized how in their experience there is a greater level of exposure of the importance of managing risk using DSDM due to high-level stakeholder involvement and collaboration. An experienced workshop facilitator noted how they bring DSDM risk management practices into traditional project environments, which reinforce any risk methodologies already in place by the client.

4.1 Risk Identification in DSDM

It became evident from all interviews that risk identification is an integral part of the risk management process. The interviewees generally spoke about three main aspects of their risk identification process, which revolved around an early suitability filter; ongoing workshops and risk log updates. One interviewee explained that before commencing a project, the suitability filter aims to identify how appropriate DSDM is for a proposed project. The existence of this suitability filter is a strong indication that DSDM clearly recognizes the potential for associated risks when using the method on incompatible projects. Furthermore the managing director noted how even in a scoping workshop they "*quickly identify risks from all stakeholder perspectives*" and later they conduct more "*formal risk assessment, which may be part of another workshop.*" This conveys how risks are considered at the very outset.

Interview participants predominantly referred to the risk log as their main medium for continuous monitoring and updating of risks. Again, one interviewee explained how *"risks would be reviewed on a weekly or fortnightly basis in conjunction with status reports that are carried out. What's outstanding on the risk log would be included in the status report."* The researcher was given access to one of the organizations risk log which contained a risk number, category, description, colour code (for high, medium or low risks), assignment of risk, date, proposed action and follow-up control. With regard to overcoming client risk, a consultant stated how "*in DSDM you actually manage scope creep with specific techniques such as prioritisation of requirements and timeboxing. Within each timebox you set the scope upfront and agree that you will not go beyond that."*

4.2 Risk Estimation in DSDM

The DSDM Consortium (1999-2003) outlines how the 'level of risk' is equal to the likelihood of its occurrence multiplied by the severity of impact, which supports literature in relation to risk estimation. The process was also evident in this case study where risks would be prioritized on this basis. An experienced workshop facilitator stated how their "*main mechanism for prioritizing risks would use a probability by impact approach.*" In contrast however, the managing director stated that they would prioritize a risk relative to the prioritized requirement to which the risk effects. The researcher concluded that while some respondents referred to the main concepts of risk estimation, the nature to which they believe risks are prioritized was extremely dependent on each interviewee's role and experience. For example, the most junior team member stated how they would "*prioritize risks and identify which are actual show-stoppers but estimating risk probabilities or allocating percentages to the impact on a project is something I haven't had experience of.*"

4.3 Risk Evaluation in DSDM

A general consensus emerged in this study that it is not considered feasible to carry out action for all risks. As a leading consultant stated "*it just wouldn't be practical and in some cases this may be more expensive to the project than what it would have to endure if the risk actually occurred.*" However, all respondents agreed on one important point when discussing responsive actions and this involved the importance of responding to the highest prioritized risks in the same manner as they would respond to the prioritized 'must-have' requirements in specific timeboxes. The researcher noted something important about this level of clarity in that the principles set down by the Consortium for requirements prioritization and timeboxing have been made applicable to the approaches adopted for managing and responding to risks.

5 Conclusions and Further Research

This research sought to discover the extent of risk management in DSDM. An analysis of the literature revealed little evidence of the extent of use of risk management practices in agile methods. It is widely recognized that agile methods themselves were introduced to combat well-known risks associated with ISD project failures such as scope creep and cost overruns. Their use of incremental development and active user involvement is an attempt to combat such risks.

Nevertheless, the findings of this research shows that the extent to which risk management is conducted in DSDM is in no way inferior to that carried out on traditional projects. While the research produced many interesting findings there is scope for further research. Given the diversity with which methods are adopted across organizations, a large-scale quantitative study may identify more generalisable themes regarding the adoption and deployment of risk management. Furthermore, the case adopted in this research focused solely on one agile method. Given the diversity that exists across the agile method family, there is a need to examine other methods.

References

1. Reifer, D.: How good are agile methods? IEEE Software 19(4), 16–18 (2002)
2. Abrahamsson, P., Warsta, J., et al.: New Directions on Agile Methods: A Comparative Analysis. In: 25th International Conference on Software Engineering, pp. 244–254. IEEE Computer Society Press, Los Alamitos (2003)
3. Vijayasarathy, L., Turk, D.: Agile Software Development: A survey of early adopters. Journal of Information Technology Management 19(2) (2008)
4. Smith, P., Pichler, R.: Agile Risks/Agile Rewards. Software Development 13(4), 50–53 (2005)
5. Hall, E.M.: Managing Risk: Methods for Software Systems Development. Addison-Wesley, Reading (1998)
6. Rowe, W.D.: An Anatomy of Risk. John Wiley & Sons Ltd., Chichester (1977)
7. Charette, R.N.: Applications Strategies for Risk Analysis. Multiscience Press Inc. (1990)
8. Coppendale, J.: Manage Risk in Product and Process Development and Avoid Unpleasant Surprises. Engineering Management Journal 5(1), 35–38 (1995)
9. Grey, S.: Practical Risk Assessment for Project Management. John Wiley & Sons Ltd., Chichester (1995)
10. Chapman, R.J.: The Effectiveness of Working Group Risk Identification and Assessment Techniques. International Journal of Project Management 16(6), 333–343 (1998)
11. Boehm, B.W.: Software Risk Management: Principles and Practices. IEEE Software 8(1), 32–41 (1991)
12. Mantel, S.J., Meredith, J.R., et al.: Project Management in Practice. John Wiley & Sons, Inc., Chichester (2001)

13. Keil, M., Cule, P.E.: A Framework for Identifying Software Project Risks. Communications of the ACM 41(11), 76–83 (1998)
14. Wiegers, K.E.: Know Your Enemy: Software Risk Management. Software Development 6(10), 38–42 (1998)
15. Chapman, C., Ward, S.: Project Risk Management: Processes, Techniques and Insights. John Wiley & Sons Ltd., Chichester (1997)
16. Charette, R.N.: Software Engineering Risk Analysis and Management. Multiscience Press (1989)
17. McManus, J.: Risk Management in Software Development Projects. Elsevier/ Butterworth-Heinemann (2004)
18. Charette, R.N.: Software Engineering Risk Analysis and Management. Multiscience Press (1989)
19. Frame, J.D.: The New Project Management: Tools for an Age of Rapid Change, Complexity, and Other Business Realities. A Wiley Company, Jossey-Bass (2002)
20. Beynon-Davies, P., Williams, M.D.: The Diffusion of Information Systems Development Methods. Journal of Strategic Information Systems 12(1), 29–46 (2003)
21. DSDM Consortium Manual Version 4.2: Risk Management (2002-2006)
22. Collis, J., Hussey, R.: Business Research. Palgrave Macmillian (2003)
23. Bonoma, T.: Case Research in Marketing: Opportunities, Problems and a Process. Journal of Marketing Research 22(2), 199–208 (1985)

A Storytest-Driven Approach to the Migration of Legacy Systems

Fabio Abbattista, Alessandro Bianchi, and Filippo Lanubile

Dipartimento di Informatica, University of Bari
Via Orabona, 4 – 70126 Bari, Italy
{fabio,bianchi,lanubile}@di.uniba.it

Abstract. In this paper, we propose an agile approach, for the migration of legacy software which combines a user story-based iterative process with automated acceptance testing. The proposed approach, named Storytest-Driven Migration (STDM), requires that acceptance tests are written both on the legacy and target versions of a software system. Because of their relevance, the quality of automated acceptance tests is assured through software inspections. As a proof of concept, we conducted a first migration project of a web application towards both a web application framework and a mobile platform.

Keywords: migration, storytest-driven development, acceptance testing.

1 Introduction

Legacy applications draw on outdated technologies but continue to be used because they serve critical business needs; in order to allow their usage in up-to-date environments migration processes are often executed [5]. When embarking on migration projects, organizations must be sure that the outputs of the target systems will be completely consistent with those of the legacy systems. Testing should be a continuous activity during a migration project; up to 80 percent of a migration project time can be spent testing the target system [3].

Storytest-driven development (STDD) [12], also called executable acceptance test-driven development [9], is an extension of test-driven development (TDD) [2]. While TDD focuses on the unit testing level, STDD starts from the functional acceptance test level. With STDD, developers write user stories first and then, in collaboration with customers, define story tests in a tabular format [11], which can be used as a living specification; for each user story they document what inputs are supplied to the system and what outputs are expected. Once story test tables have been defined, developers write test fixtures – the code which is used to take the inputs from the story test tables – exercise the system under test (SUT) and compare the observed and expected results. Finally, developers write the system code which successfully passes story tests.

We posit that organizations can benefit from adopting a test-driven approach when they have to migrate legacy applications. In particular, for the migration of legacy applications, we propose a user story-based iterative process which is driven by acceptance testing. Our approach proposes to write automated story tests both on the legacy and target versions of the application. Moreover, because of their relevance, the quality of automated story tests is assured through software inspections. We call the process *Storytest-Driven Migration* (STDM). As a proof of concept, we conducted a first migration experience of a webmail application.

The remainder of this paper is organized as follows. Section 2 describes the proposed storytest-driven approach to migrate a legacy system. Section 3 describes the first experience we conducted to validate the approach. Finally, Section 4 concludes the paper.

2 Storytest-Driven Migration

Storytest-Driven Migration (STDM) is an iterative process running over user stories and terminating when all the user stories are successfully migrated to the target platform. With STDM, user stories together with related story tests take the place of the traditional requirements documents. User stories describe the features of the legacy application as well as of the migrated application. Story tests are written to be executed both on the legacy and target versions of the application. Story tests are formally reviewed at the end of each cycle.

The process model we propose includes the following four steps, which are executed iteratively (see Fig. 1):

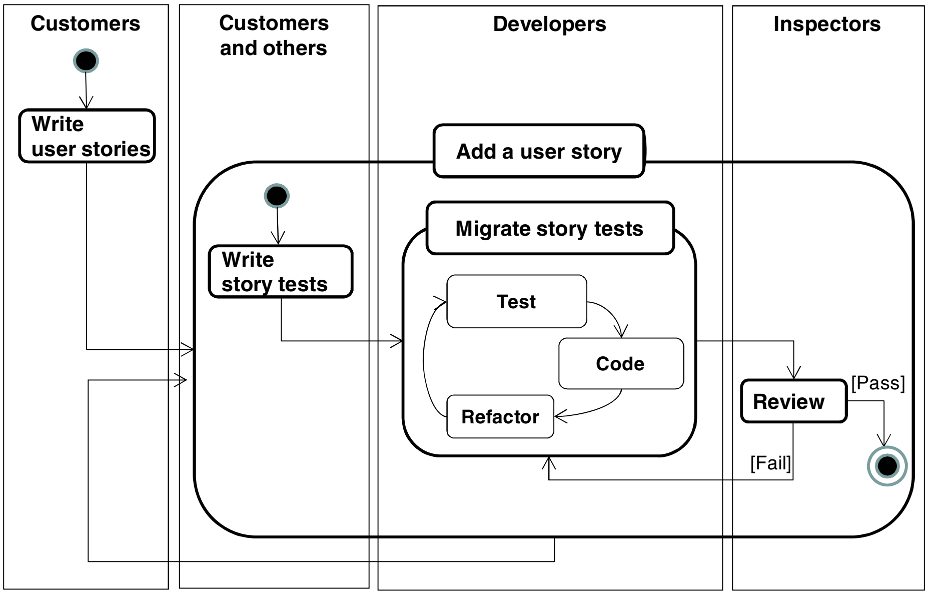


Fig. 1. Story Test-Driven Migration process

Write user stories. This first step is aimed at building a descriptive form of the requirements of the system to be migrated. In our context user stories might also consider issues related to the specific platform for which the migration is executed.

Write story tests. Once user stories are ready, customers or users together with testers or developers define story tests aimed at validating requirements described by user stories. Having the tests in place gives a clearer perspective on what one wants to achieve as well as a confidence in performing the migration. Differently to test fixtures, which are strictly related to the software platform, story tests are written just once and they should be completely or, at least, partially reused on the target system after they are written on the legacy system.

Migrate story tests. Basically, this step includes the three known activities of the TDD cycle (*test*, *code* and *refactor*) with some changes, shown in Figure 2. In the *test* step developers should write fixtures for the legacy system and run them against the legacy system. Then in the *code* step, they should migrate the legacy code, run fixtures written for the legacy system against the migrated system and, if needed, modify or rewrite fixtures according to the target platform. Finally, in the *refactor* step developers clean up the migrated code, in order to make it easier to understand. Only small changes can be applied because complete refactoring could be too expensive for the purposes of a migration project.

Review. The outputs of the previous steps, i.e., test fixtures and migrated code, are reviewed by an inspection team, and in case reworked.

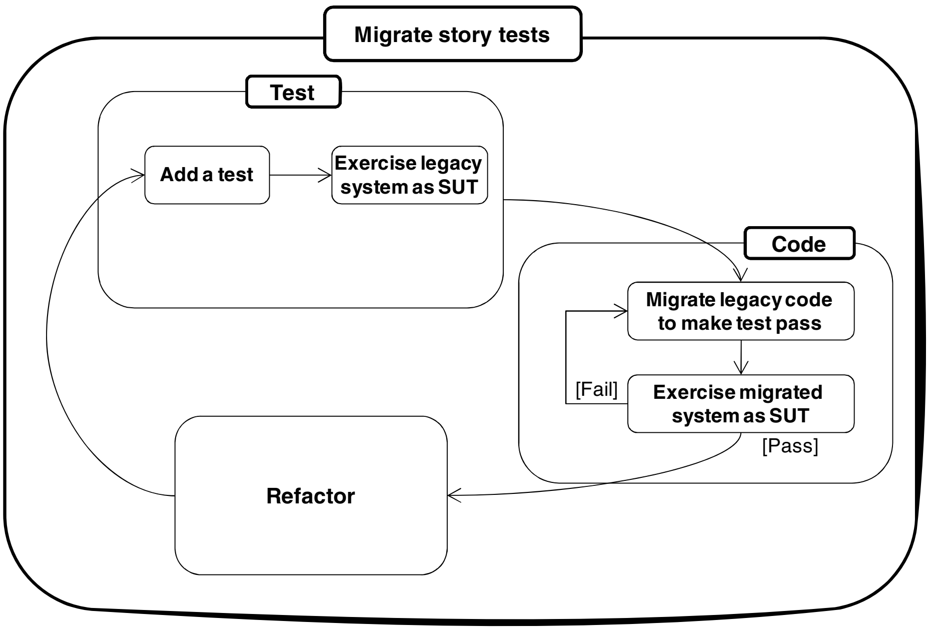

Migrate story tests
Test
Add a test
Exercise legacy system as SUT
Code
Migrate legacy code to make test pass
[Fail]
Exercise migrated system as SUT
[Pass]
Refactor

Fig. 2. Explosion of the "Migrate story tests" step

3 A First Experience with the STDM Process

The main goals of this first migration experience were the following: (1) to evaluate the feasibility of the STDM process; (2) to assess the reusability level of test fixtures written for the legacy system on the target system; and (3) to evaluate the usefulness of the selected tools to implement the STDM process.

The experience consisted of a double migration of a JSP-based webmail application [6] towards the Apache Wicket framework and Java Platform Micro Edition (Java ME). Two final-year students were involved as developers while the researchers played the role of customers.

We selected XPlanner as a project management tool and Subversion as a source code management system. We also used Google Docs for the first two steps of the STDM process (see Fig. 1). By using Google Docs, customers produced the document, including user stories, which was also accessible online to developers. Then developers, in collaboration with customers, wrote the story tests for each user story by creating different spreadsheets. Before automating the acceptance testing, developers also created FIT tables using FitNesse, according to the story tests previously defined. To implement text fixtures the developer group used JWebUnit for the legacy system, JWebUnit again for the Wicket-based target system, and J2MEUnit for the Java ME-based target system.

This first experience allows us to get some preliminary feedback, mostly based on direct observation and comments provided by the developers.

Feasibility of the STDM process. We observed that reviewing test fixtures helps to have correctly written test code which can then be used as formal specifications. According to our previous study [8] we found that inspections can improve the quality of test code. However, reviewing test fixtures when user story migration ended implied that the test issues found on the legacy are often repeated on the target system. The present study also confirms that the most common issues [10] which were found in the test code mainly affect the traceability and maintainability of test fixtures (see Fig. 3).

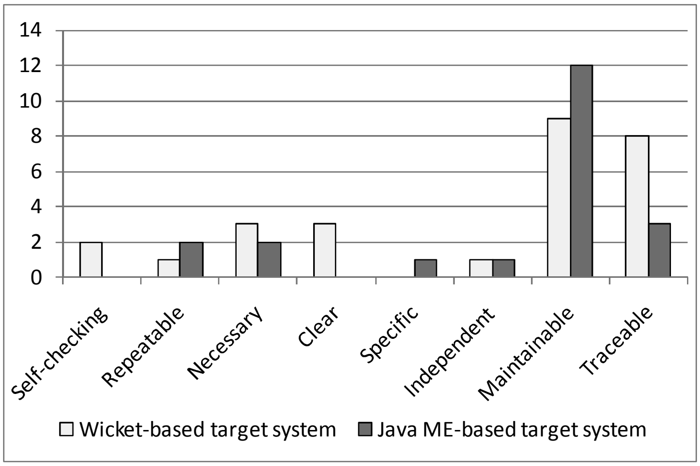


Fig. 3. Type of issues found by reviewers during inspection

Reusability of test fixtures. We found that the test fixtures written for the legacy system were almost fully reused when migrating to Wicket, while the opposite happened for the Java ME migration as the target test code had to be developed from scratch. This is because the legacy test code and the target test code for the Wicket-based solution both used the same acceptance testing framework, JWebUnit. On the contrary, the developer had to implement tests for the Java ME-based target system using a unit test framework such as J2MEUnit because, at the current date, there are no available acceptance testing frameworks for Java ME applications. However, we also found that the developer created test code faster for the Java ME-based target system than for the legacy system. Writing test code for the legacy system made comprehension of the application logic easier, and then speeded up the implementation of target test code.

Usefulness of tools. We collected both positive and negative observations about the usefulness of tools we used. Google Docs and IBIS were considered effective for the purpose they were used for. JWebUnit was found better than FitNesse to automate acceptance testing of the legacy application but there was a lack of appropriate frameworks for automating the acceptance testing of Java ME applications.

4 Discussion and Conclusions

We have presented a user-story based iterative process to migrate legacy applications. The test-driven approach has already been used for software migration [1], [4], [7], [13]. The novelty of our approach with respect to the existing literature mainly concerns two aspects.

On one side, our approach proposes to write automated acceptance tests both on the legacy and target versions of the application. In the best case we can run the same acceptance tests written for the legacy system also on the target system. This can assure that the target system preserves the behavior of the legacy system. However, since migration may also have a different platform or language as a target, test fixtures may not be reused, and thus it may require implementing acceptance tests twice. In this worst case, we posit that writing acceptance tests can bring at least two benefits: better understanding of the system to be migrated and a valid starting point to make a migration plan. Secondly, we require that story tests are reviewed by customers, especially if they take the role of specifications in place of documents. A previous study reported results about the benefits from software inspections conducted on automated unit test cases [8].

We conducted a first migration project of a web application towards both a web application framework and a mobile platform. In general, we found that iteratively migrating a legacy system worked well and that a user story can be the right portion for quick reviews. The migration from a legacy web platform to a modern web application framework, sharing the same acceptance test framework, made it possible a seamless reuse of the story test code initially written for the legacy system. However, we were not able to make any reuse of the story test code when migrating to the mobile platform. This is not a weakness of the proposed approach, but it depends on both the deep differences between the legacy and target platforms and the lack for a Java ME-based acceptance testing framework.

As future work, we intend to apply the STDM process to other and more complex legacy systems in order to assess if an earlier implementation of acceptance tests for the legacy application has a value. Through a number of experiences applying the STDM process we will define the several conditions we may encounter and some patterns to follow during migration projects.

Acknowledgments. This work is partially supported by MiUR-Italy, under grant PRIN 2006 "METAMORPHOS". We would like to thank Teresa Mallardo for the effort she put into the migration project and Mario Scalas for his valuable comments on a first draft of the paper.

References

1. Andersson, J., Bache, G., Sutton, P.: XP with Acceptance-Test Driven Development: A rewrite project for a resource optimization system. In: Marchesi, M., Succi, G. (eds.) XP 2003. LNCS, vol. 2675, pp. 180–188. Springer, Heidelberg (2003)
2. Beck, K.: Test Driven Development: By Example. Addison-Wesley, New York (2002)
3. Bisbal, J., Lawless, D., Wu, B., Grimson, J.: Legacy information systems: issues and directions. IEEE Software 16(15), 103–111 (1999)
4. Bohnet, R., Meszaros, G.: Test-Driven Porting. In: Agile Development Conference (ADC 2005), pp. 259–266. IEEE Computer Society, Los Alamitos (2005)
5. Brodie, M.L., Stonebraker, M.: Migrating Legacy Systems. Morgan Kaufmann, San Francisco (1995)
6. Brugali, D., Torchiano, M.: Software Development, Case Studies in Java. Addison Wesley, New York (2005)
7. Hennessy, M., Power, J.F.: Ensuring behavioral equivalence in test-driven porting. In: Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2006). ACM Press, New York (2006)
8. Lanubile, F., Mallardo, T.: Inspecting Automated Test Code: a Preliminary Study. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 115–122. Springer, Heidelberg (2007)
9. Melnik, G., Maurer, F.: Multiple Perspectives on Executable Acceptance Test-Driven Development. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 245–249. Springer, Heidelberg (2007)
10. Meszaros, G.: XUnit Test Patterns: Refactoring Test Code. Addison Wesley, New York (2007)
11. Mugridge, R., Cunningham, W.: Fit for Developing Software: Framework for Integrated Tests. Prentice Hall PTR, Englewood Cliffs (2005)
12. Reppert, T.: Don't Just Break Software, Make Software: How Story-Test-Driven-Development is Changing the Way QA, Customers, and Developers Work. Better Software 6(6), 18–23 (2004)
13. Varma, P., Anand, A., Pazel, D.P., Tibbitts, B.R.: NextGen eXtreme porting: structured by automation. In: ACM Symposium on Applied Computing (SAC 2005), pp. 1511–1517. ACM Press, New York (2005)

XP Practices: A Successful Tool for Increasing and Transferring Practical Knowledge in Short-Life Software Development Projects

Gabriel Tellez-Morales

University of Edinburgh, School of Informatics, Edinburgh, UK
gtellez@exatec.itesm.mx

Abstract. The Gemplus and Axalto's horizontal merge in 2006, brought several challenges, resulting in a period of general instability in the newly created company. As a result, the Gemplus Personalization Team for Latin America put in place five of the twelve Extreme Programming Practices as a tool for incrementing and transferring knowledge between the two companies and among the existing/new members of the team.

In addition to a successful knowledge transfer, results from this newly adopted approach, showed several benefits: collective code ownership, development autonomy, cleaner/more readable code, and an increment in development productivity, proving that in addition to being useful for practical knowledge transfer, XP Practices are a successful 'tool kit' to improve the software development process performance in short-life projects.

Keywords: extreme programming, knowledge transfer, software development approach, short-life software development, smart card, gemalto.

1 Knowledge Transfer and XP Adoption in the Industry

Knowledge Transfer is defined by the Institute of Knowledge Transfer [1] as *"the systems and processes by which knowledge, including technology, know-how, expertise, and skills are transferred from one party to another, leading to innovative, profitable or economic and social improvements"*. According to the Association for University Research and Industry Links [2], among the best practices to perform Knowledge Transfer, the most common/successful one is Pair Working. In the Software Industry, the action to work in pairs when coding software is known as Pair Programming -one of the most popular practices of Extreme Programming [3].

Based on its own definition, Knowledge Transfer is a key practice that top management needs to embrace among the organization as part of its most important processes to improve intercommunication, to prevent unnecessary risks (e.g. professional territoriality or ethnocentrism), or to improve the competitiveness of the company by adapting quickly and effectively when changing needs appear [4], in other words, use Knowledge Transfer to create agile organizations.

In the industry, XP has proved to be one of the most powerful techniques for generating high-quality software products in about half the time as solo programmers [5,6], to reduce the training costs of new personnel [7], to ensure more professional satisfaction & motivation, a more comfortable working environment, a noticeable productivity increment [8], and information/knowledge transfer among developers [7], helping to emphasize the developer's responsibility, and to improve the trust relation between the members of the team [9].

However, success in XP's adoption does not necessarily mean that all of the twelve practices that conform the methodology have to be implemented in a development team [3,4], but its success rely mainly on its members and their managers [4]. In consequence, its implementation is affected by several factors [7,10], being the usual ones: areas of expertise, programmers' expertise, complexity of the systems, internal conflicts, and generational differences.

2 XP Practices Implementation in a Newly Created Smart Cards Entity

2.1 Gemplus and Axalto

Gemplus was a company dedicated to the Smart Cards business, from R&D, manufacture, and personalization, to final customer services. From 2006 to 2007, I worked as a Telecom Personalization Developer for the Latin America Region; position based in the Cuernavaca Manufacturing Site where Telecom, Banking, and Public Telephony Smart Cards were printed, assembled, personalized, and packed. Gemplus horizontally merged in 2006 with Axalto (formerly Schlumberger Smart Card Department), and today the combined entity is known as Gemalto. The merge of both companies brought several challenges that needed to be solved: the incompatibility in technology and equipment, differences in corporate culture, and redundant positions; resulting in a period of general instability in the newly created company.

2.2 Overview of the Software Personalization Application Lifecycle

Gemalto is a product-based company where all information related to the product, is saved in an internal Product Data Management (PDM) tool. Data, electrical, and control (test) programs -known as the Software Personalization Application (SPA)-, are part of the product. The role of the Personalization Team was to provide its customers -Technical Consultants (TCs)- the SPA through PDM, based on software specifications given by the final customers (e.g. Mobile Network Operators) gathered by the TCs.

Smart Cards SPAs consisted in 3 parts: data, electrical, and control programs. The data program was a parser that processed the final customer input files containing thousands of rows with data to be personalized on the SIMs. The electrical program consisted in a set of Application Protocol Data Units

(APDUs) -ISO7816 communication format between the Smart Card and the off-card applications. Control programs consisted in a set of APDUs where processed data was sent to a personalized Smart Card to perform the corresponding tests.

2.3 Challenges in Gemplus' Latin America Personalization Team

With the merger, the Personalization Process for all types of Smart Cards suffered a great change for the newly created entity. Because of the incompatibility in technology and equipment, the closing-down of an Axalto's manufacturing site at Maryland, and the great range of Axalto products in stock, Management decided that Gemplus' manufacturing sites in North and Latin America would benefit in having Axalto's personalization technology on-site. As a result, the Axalto's Personalization Team (Austin, U.S.A.), needed to transfer its knowledge and its best practices in software development to the Gemplus' Personalization Teams for North America (Pennsylvania, U.S.A.) and Latin America (Cuernavaca, Mexico).

After taking the decision to transfer the existing knowledge, two developers from each one of the Gemplus' teams traveled to Austin in August 2006, with the objective to be completely familiar with Axalto's software development process, to transfer this knowledge to their own teams, to replicate the complete Axato's Unified Personalization Security Infrastructure (UPSI) at the Gemplus site where they belong to, and to be completely autonomous regarding Axalto's SPAs in a short-term time. These 4 challenges would demand more time from the existing members of the team so management came up with the idea to hire 2 additional personalization programmers. New technologies, new members, and moreover, pressure from top management to be an autonomous Personalization Team, were things that required a new approach to perform well.

2.4 Implementation of XP Practices in the Latin America Personalization Team

The use of XP practices was more emergent than planned. Typically, the software development process was neither documented nor formally defined, but rather, the manager and the developers used practices that were considered efficient before 2006. The motivation for using XP practices as a way to increase and transfer knowledge among the team, was first of all, the general uncertainty that the merger created in the general organization, and secondly, the Software Personalization Application's nature, being this: 1) the Smart Card Personalization Process involves prototype technology where the final customer requirements change rapidly, 2) the SPAs programs are small and more easily managed through informal methods requiring a minimum amount of documentation, and 3) XP's short development cycles proved to be the principal advantage while developing SPAs, constantly receiving early feedback from our customers, catching defects in an early stage of development.

Use of XP Practices

Pair programming. Pair programming was used all along the knowledge transfer project, when coding the first Axalto's SPAs, and when training newcomers in the software development process, resulting in a collective code ownership in the second half of 2007. The pairs consisted in two programmers with different level of expertise: junior/intermediate and junior/senior. Even though, the use of *homogeneous* pairs provides better results related to collaboration, using *heterogeneous* pairs is more suitable for training and knowledge transfer goals, as discussed in [7].

40-hour rule. Because of the Telecom projects' nature for Smart Cards Personalization is different from a common software programming project in the sense that the average life-cycle is 2 weeks, the present effort found only about 20 hours per week for pair programming, instead of the recommended 40-hour practice, so both programmers could handle from 2 to 4 projects per week.

Working environment. The setting for the team was designed to be a place with desks shaped specifically for programmers to sit two on a computer looking to the wall to avoid distractions. In the center, two round tables served as the setup for the support person, the on-site customer (in our case, the TC), discussions, or small meetings. Adjacent to this area was the machine used to release tested code into the production system.

Customer on-site. The TC presence in the Personalization Site for clarifying requirements was constant as XP mandates, sitting with the team, available the entire working hours to answer questions, resolve disputes, and set small scale priorities during the development of the SPA.

Small releases/iteration tracking. The PDM internal tool was used to log each release and iteration of the SPA, constantly receiving early feedback from our customers and catching possible defects in an early stage of development. The PDM tool also helped to avoid the 'cowboy' style of creating software programs by providing the project manager, the TCs, and the developers, a constant SPA's lifecycle monitor.

2.5 Results

Axalto's Unified Personalization Security Infrastructure (UPSI) was fully implemented in November 2006 at the Gemplus' Cuernavaca Personalization Site, taking a two-month time period, 2 programmers, and constant help from Austin's Team to deploy it. In January 2007, the Gemplus' Latin America Personalization Team delivered its first two projects (Telefonica Moviles Mexico and Cable & Wireless Jamaica) completely coded and tested in Axalto's technology.

In March 2007, a manager and a senior programmer from the Axalto's Personalization Team visited Cuernavaca to refine the knowledge transfer already passed to the team in the August 2006 Workshop, using pair programming for this purpose (one pair junior/senior, one pair intermediate/senior). In early April 2007 the Gemplus' Personalization Team gained its autonomy from Austin's team by taking over the complete operation the Axalto's Personalization Team was doing for the Latin America region. The new Gemalto Telecom Personalization Team for the Latin America Region was finally ready at this stage.

In the second half of 2007, the team started to create a collective code ownership in the projects where pair programming was put in place (e.g. Digicel Jamaica, Barbados, and Aruba) showing responsibility for the quality of the code when the final SPA was not working according to the specifications, resulting approximately in 90% of software specification errors done by the TC. Productivity gains were achieved through pairing because of the pair pressure effect that led to intensive sessions that discouraged frequently distractions, and the constant code review produced much cleaner and more readable code that was much easier to modify and extend by the team. Additional productivity gains were achieved by having fewer defects (almost 70%) in the SPA thanks to the short development releases, resulting in continuous feedback from the customer.

In the last quarter of 2007, the pair programming practice descended in the team. We transitioned from a process where pairing was used when writing/testing unknown code tasks were performed, into a process where pairing was conducted only when making risky changes or while debugging existing code, proving to be a successful practice for knowledge spreading, as discussed in [4].

In September 2007, the team had suffered from several changes: three new additions and one loss. The three newcomers were people with an average age of 24 years and almost no previous job experience; after applying the new practices on them, they reached autonomy in a 4-month time period. The loss of one member was because he considered the new practices to be a danger for his professional growth, although he already had 8 years in the company and he was 45 years old, proving that generational differences and internal conflicts [10] truly represent one of the problems that XP's implementation faces.

3 Conclusion

Results from the experience described in this report, indicate that the XP practices approach for incrementing and transferring knowledge between Axalto's Team and the Gemplus' Team, and among the existing/new members of the Gemplus' Team proved to be successful. In addition, the results also shown that the development process for the SPAs was more productive as similar historical projects undertaken by the members of the team. In relation to readability and maintainability of the code, this was considered to be much improved by means of constant code review, result of the pairing effect. However, results also shown that generational differences and internal conflicts were two factors that affected the XP practices implementation.

Because XP's full adoption is usually exceptional, most companies adopt XP only partially and they adapt XP to fit existing practices and philosophies [11]. As a result, the recommendation is to use XP as a 'tool kit' to improve software development techniques and increase knowledge transfer among the team [4]; recommendation, the Gemalto Telecom Personalization Team for the Latin America Region put in place when dealing with organizational uncertainty.

Although Gemalto is a successful company, it lacks a top-management definition and direction regarding software development methodologies (including documentation standards) for its Personalization Sites around the world, resulting in a customized software development methodology adoption by the development teams; customization that is common in the Telecom industry [4].

References

1. Board of the Institute. The Institute's Standards for the Accreditation of Knowledge Transfer Practitioners and Code of Professional Conduct. Institute of Knowledge Transfer (2006), `http://www.ikt.org.uk`
2. Association for University Research and Industry Links, `http://www.auril.org.uk`
3. Ramachandran, V., Shukla, A.: Circle of life, spiral of death: are XP teams following the essential practices? In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, p. 166. Springer, Heidelberg (2002)
4. Vanhanen, J., Jartti, J., Kähkönen, T.: Practical experiences of agility in the telecom industry. In: Marchesi, M., Succi, G. (eds.) XP 2003. LNCS, vol. 2675, pp. 279–287. Springer, Heidelberg (2003)
5. Williams, L.A., Kessler, R.R.: All I really need to know about pair programming I learned in kindergarten. Commun. ACM 43(5) (2000)
6. Rasmusson, J.: Strategies for introducing XP to new client sites. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, p. 45. Springer, Heidelberg (2002)
7. Arisholm, E., Gallis, H., Dyba, T., Sjoberg, D.I.K.: Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Transactions on Software Engineering 33(2), 65–86 (2007)
8. Mannaro, K., Melis, M., Marchesi, M.: Empirical analysis on the satisfaction of IT employees comparing XP practices with software development methodologies. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 166–174. Springer, Heidelberg (2004)
9. Robinson, H., Sharp, H.: The characteristics of XP teams. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 139–147. Springer, Heidelberg (2004)
10. Gittins, R., Bass, J., Hope, S.: A comparison of software development process experiences. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 231–236. Springer, Heidelberg (2004)
11. Aveling, B.: XP lite considered harmful? In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 94–103. Springer, Heidelberg (2004)

Distributed Agile Development: A Case Study of Customer Communication Challenges

Mikko Korkala[1], Minna Pikkarainen[1], and Kieran Conboy[2]

[1] VTT Technical Research Centre of Finland
P.O.Box 1100, FIN-90571, Oulu, Finland
[2] National University of Ireland Galway,
Newcastle Rd., Galway, Ireland
{Mikko.Korkala,Minna.Pikkarainen}@vtt.fi
kieran.conboy@nuigalway.ie

Abstract. The highly collaborative nature of software development emphasizes the importance of efficient communication. Agile methodologies further accentuate its importance. The importance of communication is further exacerbated in distributed environments due to temporal, geographical and cultural distances. Despite this, little is known about communication in distributed agile development. This results from the case study described in this paper suggest that the efficiency of customer communication is dependent on the nature of the actual customer relationship and the organizational policies. Weak customer relationship and deliberate information hiding may result in inefficient communication and reduced efficiency of the communication media. Thus, in order to enable meaningful communication, establishing an efficient customer relationship can be considered paramount.

1 Introduction

Effective communication is considered vital in software development (e.g. [1]), and is accentuated in agile and distributed development environments e.g. ([1-3]). Distributed software development is establishing itself as a commonplace approach in software development e.g. [3-5]. Despite the opportunities, distributed development also presents a host of challenges, and the research community has not yet developed a thorough understanding of these challenges or how they can be overcome [4, 5]. Despite the importance of customer communication in an agile context, the empirical knowledge on the subject seems to be scarce, though some evidence is available (e.g. [6]). To address this knowledge gap, we conducted a study of customer communication in a globally distributed software development project that was using an agile approach. Different communication channels were analysed based on the categorization of Cockburn [2] and the principles of Media Richness Theory [7, 8].

2 Related Literature

Several benefits have contributed to the growing interest on distributed software development, such as 24 hour, “follow the sun” development and maturation of the technical infrastructure just to name a few [5, 9]. Despite opportunities, distributed software suffers from similar problems that hinder collocated development. In general, an overall understanding of the challenges and solutions has not been achieved by the research community. [4, 5] Since the role of communication is paramount in agile development it can be argued that it is even more significant in distributed agile development. This challenge has been acknowledged and several approaches to mitigate the communication problems in distributed agile development have been provided e.g. ([3, 10]). Further research has attempted to determine the optimal balance between formal and informal communication to resolve these issues [11].

2.1 Media Richness Theory and Communication Efficiency

Media Richness Theory (MRT) [7, 8] is perhaps the most notable theory of communication, despite the criticism e.g. [12]. MRT proposes that different media are more efficient conveying different information and the selection of the medium should be aligned with the needs of the task. According to MRT information can be equivocal, thus prone to misunderstandings due to the possibility of multiple conflicting interpretations of the same content [12] or uncertainty due to a lack of information [13]. Equivocality and uncertainty of the information can be high or low [7]. Communication media capable of clarifying ambiguous subjects are considered rich, while communications requiring a long time to achieve a common understanding or that cannot clarify different perspectives are classified as lean communications [7]. Rich communication channels should be used while managing ambiguous information, while leaner channels are suitable for processing well understood messages and standard data [7]. E.g. Cockburn [2] has described his perceptions of the effectiveness of different communication media. Both the categorizations presented by MRT [8] and perceptions of Cockburn [2] are depicted in figure 1.

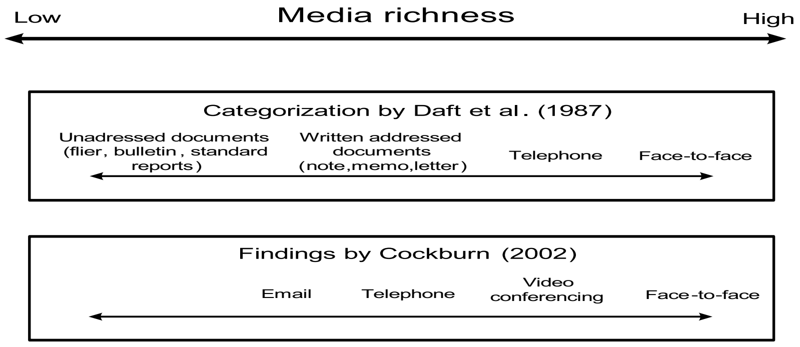

Low
Media richness
High
Categorization by Daft et al. (1987)
Unadressed documents (flier, bulletin, standard reports)
Written addressed documents (note, memo, letter)
Telephone
Face-to-face
Findings by Cockburn (2002)
Email
Telephone
Video conferencing
Face-to-face

Fig. 1. Communication effectiveness of different communication media

Face to face communication is seen as the most efficient communication media, while at the leaner end of the graph the effectiveness decreases along with the mediums capability of conveying ambiguous information. Even though Cockburn's [2] classification is not based on scientifically validated findings, it has similarities to the classification presented by Daft et al. [8].

3 Research Design

In this section, we describe the research methodology, data collection and analyzing techniques along with the research context from which we draw our conclusions. We conducted a single case study [14] in a large globally distributed organization. In this case study we interviewed a single distributed project team within the organization using a semi structured interview aiming to provide a consistent view of the customer communication environment, the communication media used, and the resulting communication challenges via qualitative data. The interviews were transcribed and analyzed using analysis approach presented in [15]. In addition, detailed field notes were taken. MRT was used in the analysis of the interview data because it provides a well defined framework and, thus, helps to focus the analysis. Communication during the requirements engineering and software implementation were key phases in which the analysis presented in this paper is focused. The case project involved two organizational units, one in the U.S. and the other in Ireland. The U.S. organization, the customer for the Irish unit, followed a traditional waterfall model, while agility was adopted by the Irish organization. There were two distinct customer representatives in the U.S. organization. The project manager considered the business analyst to be the main customer, while the Irish architect saw his U.S. counterpart as the main source of information. The project manager was responsible for management issues, while the architects communicated about technical issues. In addition, the use of various agile practices in the Irish unit was analyzed. We found that from the customer communication intensive practices *Sprint Planning, Sprints,* and *Sprint Reviews* were at use at the time of the interview.

4 Analysis and Results of the Case Study

4.1 The Use of Customer Communication Media in the Case Project

Face to face: Neither the project manager, the architect nor the team members in Ireland participated in face to face meetings with the customer organization. Face to face meetings were held only in the early phase of the project. The goal of those meetings were to define requirements for the project: *(Project manager)*: "*the first kind of cycle of those [the initial upfront design cycles] were all done on site, so the analyst would have spent a couple of weeks on site.*"

Video conferencing: Video conference meetings were occasionally used as a media for high level discussions between the project management and the customer company. Project manager: "*we would have the occasional programme meeting. So they would attend, ... but it wouldn't be a weekly occurrence*". The use of videoconferences decreased after the initial stages of the project: "*So we've probably used it [videoconferencing] more in the analysis phase than we do now on the ongoing phase.*" (*Project manager)* The architect or software developers in the team, however, did not use videoconferencing with the U.S. team or architect.

Telephone: Telephone communication was used extensively in the case project but there was a lack of teleconference meetings between the developers and the customer team. Project manager used telephone communication intensively: "*[We used the telephone] All the time. That's pretty much the standard fare."(Project manager)* The discussions focused on management topics: *"A lot of the time would be, ...some of the planning type issues. So,.... trying to get locked down on, maybe QA dates, or QA resources...*but some design issues were also discussed in some of the meetings: *"there are design discussions, in terms of, you know, our odds, where we have maybe specific technical issues where we need to bring a broader audience, we would just set up a meeting over a dial-in.*" Also the architect used the telephone regularly to communicate with the U.S. counterpart:" *"I have a conference call twice a week. I probably talk to the customer everyday."* Even though communication was active, specific issues or requirements were not solved in ad hoc manner. Instead, separate meetings were organized. Architect: "we *have .... weekly meetings, but generally meetings will be set up a couple of days in advance, talking about a specific problem or issue or requirement."* The developers were deliberately excluded from direct customer communication during the Sprints as explained by the project manager: "*actually we're trying to insulate the developers as much as possible. So, the developers would attend my weekly meeting. But really, outside of the daily stand-up, and the iteration planning meetings, and the weekly meeting, they're effectively working.*" This decision aimed to ensure that the developers could work as effectively as possible: *"So, we're trying to give them as much, you know, development time as possible." (Project manager)* The developers were not allowed to have direct communication with the customers, and the following comment indicates the reason for that: *(Architect):"No, well, that's one thing we've tried to ... because if we get too much.. I find it's a two-way thing. If you try and contact them [customers] too much from a developer perspective, they'll start contacting you guys too much (Developer): Yeah, I mean the project manager wants development team developing and not going to meetings."* Setting limits for communication is against the principles of agile development. Direct customer communication was seen as a time consuming task decreasing the time available for product development.

Email and wiki: A wiki and emails were used as an asynchronous communication channel. Wiki was the most useful communication channel in the implementation phase of the project largely due to the distributed nature of the effort. Most of the contents on the wiki were technically oriented: "*So, some of the stuff like the, high level architectural diagrams, and ... components that make up the system... are on the wiki."(project manager).* In addition higher level information was available in the wiki, as indicated: "*...you know, some of the high level planning stuff is on the wiki as well."(project manager).* The use of the wiki was however not free from problems: "*It's almost getting out of control in terms of navigation" (Developer)* Architects also used email as a decision making medium considering technical details.: *"(Interviewer): And have you had many conflicts because of some e-mail communication? (Interviewee, developer): No, not really".* It can be assumed that the technical aspects discussed with the customer were unambiguous in nature due to the apparently low level of misunderstandings. However, agreeing on technical decisions took a while: "*Architect: but I'd say most requirements, I guess they do take a while, I guess, (-) still be ongoing".* Figure 2 summarizes the use of different communication channels between the U.S. customers and Irish counterparts. The amount of use of different channels is indicated by (+) signs.

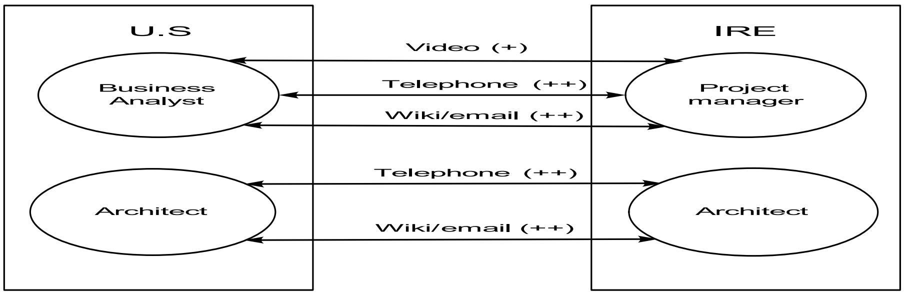


Fig. 2. The communication channels and their usage in customer communication

4.2 Customer Communication Challenges in the Case Project

Some of the challenges, of customer communication in a distributed context which appeared in the case study, are in line with the challenges presented by Ramesh [11]. The Irish organization worked with fixed up-front defined requirements which according to [11] is often the case due to the distributed software development environment. One would argue that upfront defined fixed requirements should be less ambiguous than deliberately vague agile requirements. This however did not seem to be the case in this project. *"they're not great customers, because they can just keep talking at a very high level without actually giving detailed requirements" (Architect).* The lack of detailed information of product requirements resulted into radical procedures: *"we have been coming up with our own detailed requirements, trying to understand what they need, use our experience to understand that" (Architect).* Even though telephone, a relatively rich communication medium (according to the MRT capable of resolving ambiguous issues), was extensively used among the project manager and customer group it did not seem to help developers and architects to analyse the requirements. One reason behind the situation may be the deliberate *information hiding* exercised by the customer organization. The following comment also reveals the reasons for restricting information "*Architect: It's these physical permissions [to access code implemented in U.S.],* ***but to solve those problems is a political problem****, … we're working for a part of the organization which has typically worked by themselves… we're on a different domain, different (-) directories, …so we've had some issues like that, and to join them is a very political issue. A lot of higher-up decisions need to be made that this part of that organization is going to start doing projects like this…And also because it's financed, and it's sensitive data, they always try to hide it".*

The customer company practiced process- vs. people-oriented control, rather than people- vs. process-oriented control as described by Ramesh [11], by using milestones and traditional requirements specification processes instead of continuous people driven communication. "*Interviewer: Are the customers involved [in iteration planning]? (Architect): No. They tell us, they've already told us the high-level milestones that need to be delivered*". In addition, customer relationship between the team in Ireland and the US customers was relatively weak hindering the teams from operating with a common purpose "*The customer's been quite disengaged in this project, I'd say"(Architect).* Despite the challenges, customer communication was considered sufficient, though viewpoints varied between the architect and the project manager: *"It's sufficient for the level we're at. Again, if the customer was more focused on delivery, we'd probably need more [communication]." (Architect). "as I said, we are communicating pretty much on a daily basis." (Project manager).* Despite the range of different and relatively rich communication media used in the case project (according to MRT), the abovementioned challenges remained unsolved. Thus, we claim that weak customer relationship and organizational politics that restrict information sharing may cause any communication medium to become inefficient. Instead of focusing on the selection of appropriate communication channels, the organizations should first focus on creating an efficient customer relationship and environment that enables effective communication.

5 Conclusions and Limitations

Efficient communication is one of the most essential factors in development (e.g. [1]), and is even more important in agile or distributed environments [5, 11]. We conducted a study on customer communication in a large globally distributed software development company. One of the units worked as a customer for the other organization and provided requirements for them. The customer organization was following a traditional approach while agile elements were applied in the other unit. We analyzed the use of different customer communication channels following the propositions of Media Richness Theory [7, 8] and categorization presented by Cockburn [2]. We found that several different communication channels were utilized, but the central challenges for communication were not resulting from the use of communication media themselves, but from the fixed requirements and process oriented control. We found that the customer was not involved in the implementation due to the lack of trust and the customer organization's policies hindered information sharing, thus resulting into deliberate information hiding. Detached customer not actively participating to the development and organizations unwillingness to share relevant information may result in situation in which the potential efficiency of different communication media becomes weak. Thus, organizations following distributed agile development should focus on establishing an efficient customer relationship, enabling more meaningful communication. However, the possible conflicts between the traditional and agile approaches may have had an effect on customer communications. This could be elaborated on in future research.

References

[1] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Upper Saddle River (2000)

[2] Cockburn, A.: Agile Software Development. Addison-Wesley, Indianapolis (2002)

[3] Layman, L., Williams, L., Damian, D., Bures, H.: Essential communication practices for Extreme Programming in a global software development team. Information and Software Technology 48, 781–794 (2006)

[4] Damian, D., Zowghi, D.: Requirements Engineering Challenges in Multi-site Software Development Organizations. Requirements Engineering Journal 8, 149–160 (2003)

[5] Komi-Sirviö, S., Tihinen, M.: Lessons Learned by Participants of Distributed Software Development. Knowledge and Process Management 12, 108–122 (2005)
[6] Korkala, M., Abrahamsson, P., Kyllönen, P.: A case study on the impact of customer communication on defects in agile software development. In: AGILE 2006, pp. 76–86 (2006)
[7] Daft, R.L., Lengel, R.J.: Organizational Information Requirements, Media Richness and Structural Design. Manage. Sci. 32, 554–571 (1986)
[8] Daft, R.L., Lengel, R., Trevino, L.K.: Message Equivocality, Media Selection, and Manager Performance:Implications for Information Support Systems. MIS Quarterly 11, 355–366 (1987)
[9] Gorton, I., Motwani, S.: Issues in co-operative software engineering using globally distributed teams. Information and Software Technology 38, 647–655 (1996)
[10] Kircher, M., Jain, P., Corsaro, A., Levine, D.: Distributed eXtreme programming. In: XP 2001, pp. 66–71 (2001)
[11] Ramesh, B., Cao, L., Mohan, K., Xu, P.: Can distributed software development be agile? Commun. ACM 49(10), 41–46 (2006)
[12] Dennis, A.R., Valacich, J.S.: Rethinking media richness: Towards a theory of media synchronicity. In: HICSS 1999, p. 1017 (1999)
[13] Daft, R.L., Weick, K.: Toward a Model of Organizations as Interpretation Systems. Academy of Management Review 9, 284–295 (1984)
[14] Yin, R.K.: Case Study Research Design and Methods. Sage Publications, Thousand Oaks (1994)
[15] Miles, M.B., Huberman, A.M.: Qualitative Data Analysis:An Expanded Sourcebook, 2nd edn. SAGE Publications Inc., Thousand Oaks (1994)

Customer and User Involvement in Agile Software Development

Karlheinz Kautz

Department of Informatics, Copenhagen Business School,
DK-2000 Frederiksberg, DK
Karl.Kautz@cbs.dk

Abstract. Studies of user involvement in agile development are very scarce. We provide a case study of how user involvement took place in a large agile project, which utilized the agile method eXtreme Programming. Planning games, user stories and story cards, working software and acceptance tests structured the user involvement. We found genuine customer and user involvement in the form of direct and indirect participation in the project. The involved customer representatives played informative, consultative and participative roles in the project. This lead to their functional empowerment i.e. the users were enabled to carry out their work to their own satisfaction and in an effective, efficient and economical manner.

Keywords: Agile software development, user involvement, practice study.

1 Introduction

Agile software development (ASD) insists on the customer taking control and being constantly involved and stresses a collaborative partnership based on daily interaction between developers and the customer [1]. There are however only a few, empirically sound studies on ASD, and even less on user involvement in ASD [2]. These studies of agile practice have shown that customer representatives might have decision power, but only a limited understanding of the users' needs, they might not be the actual users of the software to be developed, who in turn might have the necessary knowledge, but not the authority to decide on system features [2]. Users rarely take the role of the customers [4]. The customer role might even been carried out by substitutes from the development organization such as product managers or marketing staff [5]. This is the background for our research, which attempts to answer the question how and with which effect customers and users participate in agile development and design activities in practice.

2 Research Method

The research is qualitative. It is based on an empirical case study of a commercial agile development project in a large German public sector organization, called WaterWorks

(WW), performed by a software company, called AD. The data was collected in 12 semi-structured, open-ended interviews which included nearly a third of the project team. The interviews were tape-recorded and subsequently transcribed. For the data analysis a software tool (NVIVO7) was used. The interview data was supplemented with company and project documents. The data collection, coding of the data, and data analysis were guided by the 4 value pairs underlying agile development: (1) individuals and interactions over processes and tools; (2) working software over comprehensive documentation; (3) customer collaboration over contract negotiation; and (4) responding to change over following a plan. The data was in particular analyzed with regard to customer and user involvement.

3 The Case Setting: The OMS Project

The project was concerned with the development of an operations management system (OMS). The system was developed with a graphical user interface and a backend to interface the technical infrastructure as defined by an underlying ERP system. The project was organized in 4 subprojects to provide support ranging from customer management to the maintenance of a duct system. At the time of the project AD consisted of about 25 employees, 20 of them being developers, and based its development approach on XP [6]. The formalized method includes planning techniques called planning games, user stories and story cards to specify user requirements, onsite customers to support customer-developer communication, daily, stand-up meetings of the project team to support team communication, pair programming, re-factoring, collective ownership, continuous integration and testing to develop the software proper and tuning workshops to improve the development processes regularly. AD extended the method with some project management processes to cater for larger projects such an overall project plan, formal reporting mechanisms and a formal contract based on a requirements specification called realization concept, which had been produced by the customer. The project was organized in 2 phases. In a first 12 months exploration phase prototypes catching requirements and possible solutions were developed. This led to the development of the realization concept by the customer organization and their decision to contract AD also for the development of the OMS proper. In this main development phase a team of about 12 development staff with multiple roles such as project manager (PM), subproject manager (SPM), analyst, customer contact, and developer worked onsite in a WW building. A sophisticated management structure with one SPM acting as contact person from AD and one acting as onsite customer from WW for each of the subprojects was, in addition to 2 overall project managers, 1 from each company, established. The WW SPMs and onsite customers were managers and team leaders in operational divisions and as such also OMS users. They were by and large, however not the whole time onsite. The project also comprised a varying number of other users, representing operational staff from different divisions. They were mostly actively involved in feedback and testing activities. When this study was performed phase two had been going on for 4 months. Responding to an inquiry during our analysis AD stated that the project ended 10 months later on time and budget with all parts of OMS being operational.

4 An Analysis of the OMS Project

The project was described by both the customer and the supplier as a success. One user based on his experience, had become a full member of the development team. Otherwise, customer and user involvement took place on an ongoing basis; the planning games; story cards, the working software as well as the acceptance tests structured the continuous day-to-day-contacts, communication and collaboration.

4.1 Planning Games, User Stories and Story Cards

At the start of phase two a number of different documents existed, which were all comparable short and concise. The planning games at the beginning of each iteration were based on the overall realization concept and requirements lists. These were largely produced by the AD PM and some of the AD SPMs. They developed these documents with input from the onsite customers. The story cards were solely produced and estimated by the developers. The developers and the customers then together prioritised these cards. In this context a WW SPM explained his own role: *"We are communicators to back up the kind of system we'll get. My management has put me here 100%, and when I get a call or an email, I show up and then we just discuss the matter."* Another WW SPM supplemented: *"But it's not just that the SPM develops his requirements at his own discretion, he holds a strong contact with the people from his division. He, quasi, sucks the requirements out of the division and carries them into the project. He then has to prioritize and has to look whether this is in his budget."* An AD SPM talked about the difficulties of converting the requirements into design (*"it's not easy to find out from the WW people what they want")* and declared that design is the task of the developers. However, the design is always developed with close participation of the WW SPMs and other users and always under the mandate of the WW SPMs. Another AD SPM went even further and said: *"Somehow they develop the tool themselves; we have not developed anything, which we invented ourselves and we always communicated to all."* This form of customer collaboration apparently provided some structure to cope with the complexities of a comparatively large ASD project, while leaving room for less structured, but necessary collaboration as well. That is to say, when implementing the story cards, it became obvious that some additional collaboration was needed. One WW SPM estimated that contact with the onsite customer was necessary for nearly every story card. He put forward that maybe two thirds of a card's contents was clear and said *" 60% are there and 40% have to be directly coordinated, 20% in the middle of the iteration and 20% in the end, the users look at it and then they find out that their requirements weren't understood the way they meant it, but that's normal."* This illustrates the importance of the working software for the development and design process.

4.2 Working Software

The presentations of working software were identified as another basis for customer and user involvement. In the project a first software release was provided after 3 months with the others to be delivered every 3-6 months. Each release was organized in iterations of 3-6 weeks duration. The AD PM described how the working software, which was produced story card by story card, attracted the WW SPMs and how they seamlessly participated in the development process: *"Some of the SPMs are here nearly on a daily basis and we have the weekly meeting; we can already before an iteration is released fix things, and we show them and have peace."* The close involvement needed getting used to as one AD SPM expressed: *"As a developer one is not that used to that one sits at the customer site and has a customer who passes by every day, there are moments where this is disturbing."* Feedback about and change requests for the software design were brought forward by the onsite customers in weekly feedback loops, which were built into an iteration. The AD PM explained: *"And then after a week the customer rep is back and wants to see what happened during the week and he gets the information and provides first feedback."* This had the following consequence: *"Often we show the customer rep something once a week and then he's going 'well, I thought this would be different'. Thus there are always small changes."* as one developer put it. As presentations to one onsite customer were not considered sufficient, the working software was also presented to larger groups of prospective users. The AD PM stated: *"Well, at latest when an iteration is finished, sometimes already in the middle presentations are run for users. Not always in front of many users, but the customer SPM gets some people together and says: 'Here, look, do we develop in the right direction?'"* In addition, the onsite customer representatives regularly performed 'road shows' with the working software in the user departments to collect feedback and ideas and proposals for improvements. The AD SPMs also sought direct contact with the users and one of them reported: *"Then I seated myself for two weeks in the duct operation station to look how well the software actually fitted the operation."* These frequent feedback loops had the effect that minor misunderstandings were caught and dealt with as changes early before they could grow into something larger. But the working software also brought to light some problems of participatory design as according to one AD PM: *"We could not understand where the discontentedness of the customers' SPMs came from until we found out that they had an expectation that we should do more than we said, they have the attitude 'here you have my problems, surprise me' ... but that's not how it is done here."*

4.3 Acceptance Tests

The AD PM explained that *"Between two iterations there is always a test phase, which is a post activity of the preceding iteration and there is a concept phase, which is the preceding activity of the next phase."* He confirmed that the acceptance tests were run by the customer, meaning that the customer had the responsibility and decision power in these tests. The tests were performed according to a protocol and *"they always comprise end users"*. Before an acceptance test was performed less formal preparations took place often triggered by a WW SPM. A typical acceptance test was portrayed by some AD SPMs: *"... a test lasts just one day ... then two or three divisional managers and other people, who own the task and who have to work with it, are present and they test it then." "I note down the bugs on a bug list, monitoring the acceptance test otherwise is the responsibility of the WW." "During the acceptance test ... no, the developers usually do not participate in those."* There were however exceptions from this rule and in some cases also developers participated in thetests.

The results of these tests were described by the one AD SPM as *"How many mistakes we found, but we got quite some encouraging feedback; first of all they could imagine to work with it, they liked it better then the [ERP-based solution]."* The WW SPMs confirmed this approach and said: *"Well, we have the acceptance tests before an iteration is approved." "The iterations help us to get in touch with the users and get something tangible." "The test phases are decisive …and they are coordinated with the staff council."* By and large they were content with this form of participation and considered themselves as part of the development.

5 Discussion and Conclusion

We base the further discussion of our case on concepts introduced in three writings. Clement [7] argues that the purpose of user participation is user empowerment. He distinguishes between functional and democratic empowerment. The former means that the users should be able to carry out their work to their own satisfaction and in an effective, efficient and economical manner. Their participation in the design process supports this objective. Democratic empowerment means that they should have the mandate to participate in decision-making. We can determine that functional empowerment has been achieved. All stakeholder groups reported that they were content with the project outcome. We have no evidence for any democratic empowerment other than the staff council's peripheral involvement and the onsite customers' authority to take decisions concerning their staff's and their own workplaces. Mumford [8] classifies two different forms of involvement, namely direct and indirect user participation, where the user is represented by some kind of intermediary. Direct and indirect participation are defined through the users' direct participation in the project (team) or their direct or indirect contact with project staff from the development organization. We found both direct and indirect participation: the onsite customers were WW staff, who themselves would work with the future system to a certain extent: they exerted direct participation. They were also intermediaries for other users, f. ex. operative staff in the customer division and the duct net division. These employees indirectly participated when they commented during presentations or when they provided their viewpoints or descriptions of their work processes to the onsite customers. They have however also participated directly when they tested the results of the iterations or when they were observed or conferred the developers, who were in contact with them directly in the divisions as f. ex. the developer, who visited and stayed in the duct net division while the result of an iteration was in operation. Damodaran [9] differentiates three user roles in the design and development process. The user can play an informative, consultative or participative role. As informants users merely provide information about their work and might be the objects of some observation. In a consultative role they are asked to comment on preset design solutions. In a participative role they actively participate in the design process and have decision-making power regarding the solution. Although one of the WW SPMs went so far to report to management that in the project only things, which the employees wanted, were done, a more differentiated picture appears when we analyze the different user roles. Those users in the divisions, who were not also onsite customers, had the role of informative or consultative users. They provided both the onsite customers and the development staff at AD with information about their work, their needs and their preferences. Some of them were observed during their work by some developers and they provided their comments during presentations and tests. Although this information and comments were taken into account and many of them were realized, it goes too far to consider this a participative role as it were solely the onsite customers, who had a decision making mandate. They, thus, as so far as their positions as (a minority among the) future users were concerned, were clearly in a participative role, although they f. ex. did not themselves write the user stories and the story cards. However, no design decision could be taken without their agreement. Beyond their participative role, the onsite customers also had informative and consultative roles.

In summary, our case description and analysis contributes to an understanding of user involvement in agile development in practice. Prior research has shown that end users rarely take the role of the customers, sometimes the customer is even represented by a development organization's own staff [3,4,5]. In contrast, we found genuine customer and user involvement in a successful agile development project. While further academic research into user involvement in ASD is necessary, our study might serve as an inspiration for practitioners, who want to firmly establish user involvement in ASD in practice.

Acknowledgements. I like to thank R. Klischewski, who collected the data with me, S. Matook, who was involved in the analysis and H. Sharp, who provided literature, which documents empirical studies of user involvement in agile development.

References

1. Highsmith, J.: Agile Software Development Ecosystems. Addison-Wesley, Boston (2002)
2. Dybå, T., Dingsøyr, T.: Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology 50 (2008), doi:10.1016/j.infsof.2008.01.006
3. Robinson, R., Sharp, H.: The Social Side of Technical Practices. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 100–108. Springer, Heidelberg (2005)
4. Martin, A., Biddle, R., Noble, J.: The XP Customer Role in Practice: Three Studies. In: Proceedings of the 2nd Agile Development Conference, Salt Lake City, USA (2004)
5. Robinson, H., Sharp, H.: XP Culture: Why the twelve practices both are and are not the most significant thing. In: Proceedings of the Agile Development Conference, Salt Lake City, USA (2003)
6. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison Wesley Professional, Boston (2004)
7. Clement, A.: Computing at Work: Empowering Action By 'Low-level' Users. Communications of the ACM 37(1), 52–63 (1994)
8. Mumford, E.: Designing Human Systems for New Technology - The ETHICS Method, Manchester. Manchester Business School, UK (1983)
9. Damodaran, L.: User involvement in the systems design process – a practical guide for users. Behaviour and Information Technology 15(6), 363–377 (1996)

Integration of Extreme Programming and User-Centered Design: Lessons Learned[⋆]

Zahid Hussain[1], Harald Milchrahm[1], Sara Shahzad[1], Wolfgang Slany[1], Manfred Tscheligi[2], and Peter Wolkerstorfer[2]

[1] Institute for Software Technology, Technical University Graz, Austria
zhussain@ist.tugraz.at
[2] CURE - Center for Usability Research & Engineering, Austria
wolkerstorfer@cure.at

Abstract. One of the most important factors for the success of a software application is user acceptance by having a usable user interface. Since summer 2007 in our project regarding mobile phone application, we have combined Extreme Programming and User-Centered Design methodologies aiming to deliver usable and useful software. The HCI instruments we have integrated are: user studies, personas, usability expert evaluations, usability tests, automated usability evaluations in the form of extended unit tests, as well as lightweight prototypes. After one and half years we conducted a retrospective full-day workshop with our off-site usability engineer to reflect on the adopted process regarding the HCI instruments. This paper presents those reflections - the lessons that we learned.

Keywords: Agile Methods, Extreme Programming, Usability, User-Centered Design, Mobile Application.

1 Introduction

One of the most important factors for the success of a software application is user acceptance by having a usable user interface. Extreme Programming (XP) - one of the mostly adopted processes of agile methods in the industry - aims to continuously deliver quality software by satisfying the customer. The Agile Manifesto does not clearly mention that the customer should be an end-user and rarely end-user takes the customer role [1]. There is also the evidence that coordination only with the customer does not ensure good usability [2]; which can result in lowering the user acceptance rate. User-Centered Design (UCD) is an approach to user interface design focusing on end-users throughout the planning, design, and development stages of a product [3].

[⋆] The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

Recently there has been an increasing interest in integrating agile and user experience/UCD methodologies both in the agile community and the HCI community. In fact there were special tracks and dedicated workshops in the Agile 2008 conference[1] as well as in the CHI 2008 conference[2]. Being integrated into agile methods, UCD/usability engineering helps to reduce the risk of running into wrong design decisions by involving real users, and results in increasing the acceptance of software applications [4]. We have integrated XP and UCD in our project regarding a multimedia streaming application for mobile phones since summer 2007. The end-users are indirectly involved in the process by our use of different HCI instruments like user studies, personas, usability expert evaluations, usability tests, extended unit-tests, and lightweight prototypes [5][6]. Recently a one-day retrospective workshop was conducted which was attended by all the team members and the usability engineer to reflect on the integrated process as well as on the HCI instruments after introducing them at the start of our project. This paper mainly describes those lessons learned. The next section describes the related work of combining Agile/XP with UCD methodologies. Section 3 describes the project context. Section 4 gives details about the retrospective reflection workshop regarding the HCI instruments used and the adopted process. Section 5 concludes the paper with future work.

2 Related Work

The integration of agile methods with HCI practices was discussed by Kent Beck and Alan Cooper in 2002, concluding that both interaction design and XP have strengths to be combined [7]. There are several studies examining various aspects of the integration of both methodologies. Patton [8] has described the way of incorporating interaction design in an agile process. Chamberlain et al. [1] have conducted an ethnographic field study to explore a framework for integrating agile methods with UCD. In their case study, McInerney and Maurer [9] interviewed three UCD specialists for integrating UCD within agile methods and reported positive feedback. Ferreira et al. [10] investigated several projects for the relation of UI design and agile methods. Fox et al. [11] also conducted a qualitative study that describes how the agile methods and UCD are integrated in industry. Obendorf and Finck [12] report their experience of combining XP and scenario-based usability engineering. The HCI instruments that we have used in our project and the integrated process are described below.

3 The Project Context

Since summer 2007, we are working on a project to develop a multimedia streaming application for mobile phones. The project is based in Austria and will end in 2010. The team consists of six full-time regular members, five developers and

[1] `http://www.agile2008.org/`
[2] `http://www.chi2008.org/`

a product manager who plays the role of the “on-site customer”. One dedicated off-site usability engineer of a partner usability research center is also included in the team regarding usability guidance. The application enables a user to perform content-based search for audio and video content in large digital archives and play it on a mobile phone [6].

4 Retrospective Workshop

Subsections below describe the HCI instruments and the reflections about them discussed in the retrospective workshop.

4.1 User Studies

User studies are the instrument for getting knowledge about end-users. The purpose of user studies is to uncover user needs, desires and contexts of use. In an agile UCD process they can be used to develop new or refactor the existing personas as well as in the user-story creation process. In our process we employed user studies in the form of laddering interviews and field studies. The laddering interviews were conducted in autumn 2007 and the results were published in [13]. Currently, a large field trial study is being conducted with 150 real end-users spread throughout Austria. The trial study includes field trials with diary studies, contextual interviews, laboratory usability tests, questionnaires, and focus group.

4.2 Personas

Personas are archetypical figures - fictitious characters created as a tool to represent a typical user group. In our process, initial personas were created based on initial user studies and would be iteratively refactored when the new user studies suggest some changes. The personas helped to gear the project towards the on-site customer and end-users. However, the initially developed personas were not satisfactorily distributed to either the development team or the customer-on-site by the usability engineer. In addition to that, it was the fault of the development team and the on-site customer because they did not give much credit to the two personas which were provided. Nevertheless they got in touch with them instead of neglecting them fully. It was concluded that personas should be properly introduced to the team so that they will be present consciously or unconsciously in the minds of the team members during planning, developing or undertaking any decision process.

4.3 Lightweight Prototypes

We make use of two different types of mock-ups; low fidelity paper mock-ups and high fidelity mock-ups and get them evaluated by the customer. As both the developers and the customer have been increasingly gaining knowledge about usability engineering, evaluating paper prototypes with the customer is good on one hand but on the other hand the customer has now become an expert user instead of a casual or novice user. So there is always a chance to ignore the actual needs of real casual end-users. We have mitigated this risk by having more ad hoc input from the usability engineer and suggested to conduct a formal usability test with at least 10 end-users after every release, i.e., quarterly. For special "ad hoc" questions instant messaging is used to gather HCI feedback within a short time frame from the usability engineer regarding stories or mock-ups. It would have been more beneficial if the usability engineer would have been present on-site to quickly give his feedback, and instead of developers he should do the prototyping not only with the customer but also with at least a few end-users.

4.4 Usability Expert Evaluations

Expert usability evaluations are reviews conducted by experts. In our project, usability expert evaluations by the off-site usability engineer are given by instant messaging, email, and video-conferencing; usually in the form of an ad hoc input. In our project usability input is needed at different timings: when writing UI related stories and before or during implementation of the stories, as well as after implementation. As the customer writes stories with the help of a developer, so it is decided that when UI related story is written, it should be sent along with the refined paper prototype to the usability engineer at least three days before the iteration planning. The advantage is that during the iteration planning you already has the usability tested story. When technical questions arise during the implementation - for instance that a certain demand from the usability side would cost too much, it is advised to call the usability-engineer or have a short video-conference. As a result you should get far less usability-fixes to make. For us two hours to one day duration is perfect for this quick feedback of the usability engineer. After implementation (when it is deployed) it can be delayed for days to get usability feedback. It was suggested that the usability engineer should give his feedback in the form of stories along with wireframes when he thinks they are needed, so he should be trained in writing stories with the help of one of the developers.

4.5 Usability Tests

Usability tests involve real users testing an application. We conducted a formal usability test with 10 users in January 2008 which was also attended by two developers as observers [6]. It was noticed that the mindset of developers changed dramatically when seeing real users handling the application as they observed the users, the developers who attended the usability test got more biased towards user-centered thinking than the others. When it comes to the results of the test there was an agreement that the test was too early in the project to tell us a lot about the usability problems of the application as the system was very fast moving at that time, changes in features were high because of new demands from the stakeholders. Furthermore the reporting period was too long. Until the report arrived, the application had so much changed that the recommendations were partially obsolete. Therefore, smaller tests after every 3 iterations were recommended. Since the system is a very fast moving target, not always the entire system should be tested. Big formal tests were recommended after every release when major changes are expected.

4.6 Extended Unit Tests

Extended unit tests root in automated usability evaluation. Our intended approach extends the XP unit tests by adding usability-specific test cases. Code based tests are enhanced with semantics to achieve this goal. For example code based tests can check against guidelines like the usage of capital letters on buttons or the correct label of a button. So far we haven't focused on extended unit tests because of priorities to other areas but intend to work in this direction from the second quarter of this year.

5 Discussion and Conclusion

Since summer 2007, the mentioned HCI instruments have been used in our project for enhancing the usability of the product. Until now we have learned few lessons that are summarized here: In our project the XP process fits well into the UCD approach because of the many overlapping principles (focus on delivering value, iterative development, end-user incorporation, continuous testing) of both methodologies [6]. The usability engineer is well integrated into the development team. We found no cultural difference until now. Developers have gained insights into the UCD practices, while the usability engineer learned the origin of usability problems [5].

Furthermore it was found out that especially ad hoc input can be given sufficiently via mail, since most of the time no synchronous communication between the project members is needed. It is also fitting for various response times since for example, for usability input in the story-writing process it is sufficient to get results within 3 - 4 days. When quick fixes are needed or other input during an urgent re-planning, the usability engineer should be easy to contact for a quick advice via cell-phone or chat. The interaction with usability engineer early in the story creation-process results in saving time, increasing motivation, and gaining better realization of needed usability input early in the development. Furthermore, instead of a big report of a formal usability test, the usability engineer should give report in the form of checkpoints which then be converted into stories quickly. Usability engineer should be trained in XP-story writing to be able to deliver the user-stories in a technical-aware manner. Proper customer and usability engineer coordination is necessary for enabling a good usability process in the development.

The field trial study is underway these days. We have implemented user-tracking and feedback mechanisms already in the basic architecture. The interviews, diary studies, usability tests, focus group, and log file analysis results will provide feedback on not only how the end-users perceive the product but it will also allow to collect statistical data from their actual usage behavior. In this way we will be able to provide more insights about the integration of HCI instruments into our adopted process. We will also gain insights into the context of mobile multimedia usage. We aim to continuously optimize our process till the project ends and will share the knowledge gained to the agile as well as the HCI communities.

References

1. Chamberlain, S., Sharp, H., Maiden, N.: Towards a framework for integrating agile development and user-centred design. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 143–153. Springer, Heidelberg (2006)
2. Jokela, T., Abrahamsson, P.: Usability assessment of an extreme programming project: Close co-operation with the customer does not equal to good usability. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 393–407. Springer, Heidelberg (2004)
3. W3C: Notes on user centered design process (UCD) (2004), `http://www.w3.org/WAI/EO/2003/ucd` (Last visited:19.01.2009)
4. Memmel, T., Reiterer, H., Holzinger, A.: Agile methods and visual specification in software development: A chance to ensure universal access. In: Stephanidis, C. (ed.) HCI 2007. LNCS, vol. 4554, pp. 453–462. Springer, Heidelberg (2007)
5. Wolkerstorfer, P., Tscheligi, M., Sefelin, R., Milchrahm, H., Hussain, Z., Lechner, M., Shahzad, S.: Probing an agile usability process. In: CHI 2008: human factors in computing systems, pp. 2151–2158. ACM, New York (2008)
6. Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., Wolkerstorfer, P.: Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application. In: USAB 2008. LNCS, vol. 5298, pp. 313–330. Springer, Heidelberg (2008)
7. Nelson, E.: Extreme programming vs. interaction design. FTP Online (2002)
8. Patton, J.: Hitting the target: adding interaction design to agile software development. In: OOPSLA 2002 Practitioners Reports. ACM, Washington (2002)
9. McInerney, P., Maurer, F.: UCD in agile projects: dream team or odd couple? Interactions 12(6), 19–23 (2005)
10. Ferreira, J., Noble, J., Biddle, R.: Agile development iterations and UI design. In: Agile 2007, pp. 50–58. IEEE Computer Society, Los Alamitos (2007)
11. Fox, D., Sillito, J., Maurer, F.: Agile methods and User-Centered design: How these two methodologies are being successfully integrated in industry. In: Agile, 2008. AGILE 2008. Conference, pp. 63–72 (2008)
12. Obendorf, H., Finck, M.: Scenario-based usability engineering techniques in agile development processes. In: CHI 2008, pp. 2159–2166. ACM, New York (2008)
13. Leitner, M., Wolkerstorfer, P., Sefelin, R., Tscheligi, M.: Mobile multimedia: identifying user values using the means-end theory. In: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services, pp. 167–175. ACM, Amsterdam (2008)

Optimizing Agile Processes by Early Identification of Hidden Requirements

Agustín Yagüe, Pilar Rodríguez, and Juan Garbajosa

Technical University of Madrid (UPM)
SYST Research Group
E.U. Informatica. Ctra. Valencia Km. 7. E-28031 Madrid
agustin.yague@upm.es, prodriguez@syst.eui.upm.es, jgs@eui.upm.es
http://syst.eui.upm.es

Abstract. In recent years, Agile methodologies have increased their relevance in software development, through the application of different testing techniques like unit or acceptance testing. Tests play in agile methodologies a similar role that in waterfall process models: check conformance. Nevertheless the scenario is not the same The contribution of this paper is to explain how the process can be modified to do early identification of hidden requirements (HR) using testing techniques in agile methodologies, specifically using failed tests. The result is an optimized agile process where it may be possible to reach the desired level of functionality in less iterations, but with a similar level of quality. Furthermore it might be necessary to re-think process elements role, e.g. tests, in the Agile context not assuming waterfall definition and scope.

1 Introduction

[1]Software industry today is characterized by a continuous dynamism and variability [1] where time to market is more and more an essential constraint to be respected. In this environment, agile methodologies have appeared to adapt industry practices to current market changes. Agile approaches are able to provide fast responses, because they are based on continuous integration, integrated testing, incremental delivery, together with teams socially structured to embrace change. One of the most evident measures of success of agile software development is the extent the obtained product meets end-user needs. The artifact widely used to represent system requirements are User Stories (US). A key aspect in Agile is a fast communication between customers and developers[2], however some problems related to requirements remain [3,4].

Waterfall methodologies have a common an exhaustive requirements engineering process. The soundness of the approach for product development depends on how complete and consistent the set of identified requirements is at the end of the requirements analysis [5]. In Agile methodologies, the list of business requirements does not need to be complete at the beginning of the project and some of

[1] This work has been partially sponsored by OVAL/PM TIN2006-14840 (MEC,Spain) and FLEXI FIT-340005-2007-37 (ITEA2 6022) (MINER, Spain).

them come out during the development and to update the list of requirements. The use of failed execution tests to identify some hidden customer requirements was pointed out in [6]. A test failure is not always produced by an error in the design or coding; it may show the evidence that some assumed requirements had not been considered for not being *visible* [7]. There are some publications related with to requirement interaction such as [8]. The requirement that has not been considered, and that causes the test failure, will be one that interacts with another already considered. This paper is based on the scenario analysis detection method presented in [9]. The main goal of this paper is to propose a process that includes the analysis of failed tests to search for hidden requirements. This result is an optimized agile process since it may be possible to reach the desired level of functionality in a lower number of iterations, but with a similar level of quality. The optimization is based on three factors: time, functionality and resources. In our approach it is possible to reduce time required to reach a similar level of functionality, maintaining the resources and with a similar level of quality. Other alternatives would be also possible.

The remainder of this paper is organized as follows: section 2 describes in brief the related works identified about agile requirements management and agile testing techniques. Section 3 analyzes the role of requirements process and testing tasks in eXtreme Programming and our process improvement approach is presented. Section 4 describes a case study with a discussion of our approach. Finally, conclusions and future work are presented in the last section.

2 Background and Related Work

The activities related to customer needs identification and management are some of the most critical and complex in software engineering in Agile methodologies [10]. Problems coming up at the stage of identifying customer needs have a negative impact on the resulting product and could compromise the advantages that offer Agile methodologies. Ways to improve requirements related practices were [11] and still are needed. In particular, Boness and Harrison [12] describe a technique called *Goal Sketching* that can be used as part of Agile processes to improve requirements engineering in the way to represent project goals. Also, Grünbacher et al. [3] present a work focused on XP practices [13] proposing the inclusion of requirements negotiation techniques such as *EasyWinWin* to help stakeholder to discover, elaborate and negotiate software requirements; it does not pay special attention to the testing. Some other works propose the use of use cases[14], or scenarios and aspects[15], but they are not focused on the process of requirements discovery that are assumed implicitly by customer. However, most of studies are focused on the management of already identified requirements, but not on providing mechanisms to facilitate the elicitation of needs not identified at beginning of project (*hidden requirements*) as is the case in this study. E.g., [16] is focused on requirements elicitation when a customer is not available *in situ* but not on the customer assumed needs.

In Agile methodologies, testing is a fundamental practice. Many works such as [17,18,19,20,21] show this. However, they are not focused on using tests to identify hidden requirements. This possibility was highlighted in [6]. This paper, we go a step ahead describing a first step for a systematic way to identify if hidden requirements are the reason for a failed test in agile methods, and to discover these requirements. This may result in an improvement of agile processes.

3 Elements in Early Identification of Hidden Requirements

The starting point of Agile projects is the *Product Backlog* that represents the capabilities that should be implemented. Requirements are obtained in an initial process called *Planning game* including two main phases: *Exploration Phase* and *Iteration Planning.* The *Product Backlog* (PB) lets the XP team to establish an prioritized initial plan and provides the XP team with a project overview. This phase should be as short as possible. XP considers that requirements might be changing over the software project execution by many different reasons: customer perception, environment, technology, or new market opportunities. The next step is to organize user stories in sprints or iterations. As iterations are executed during a limited and short period of time, the feedback from each new sprint is received also after a short period of time. This let the XP team discover very early if the system is being developed as it was expected by the customer or not. At the end of each iteration the XP team and the customer review the development done looking for new functionalities that are added to the PB.

For Test Driven Development [17] the sequence of actions is: first, user stories description; then for each user story, acceptance tests definition; after that implementation to satisfy the acceptance tests; finally acceptance tests are run, if possible by automated testing tools. Underlying the process is continuous integration environments facilitating the validation of small product increments. As tests are defined directly from requirements, e.g. user stories, any success in tests represent that some requirements have been achieved. The main testing techniques applied are: unit testing, focused on source code, and acceptance testing derived from acceptance criteria. The automation process at this level is more complex because it requires specific languages to write acceptance tests. In relation with testing levels [17,22], some development approaches have been defined; e.g., TDD is addressed by unit testing, ATDD is addressed by acceptance tests or STDD that is addressed by User Stories. Each testing technique

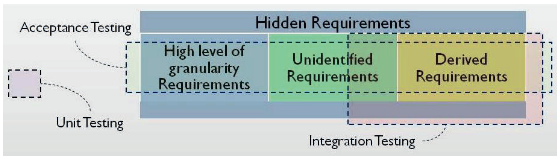


Fig. 1. Identified Requirements

plays a different role in the identification of hidden requirements. Unit testing is not very useful to identify hidden requirements. In the opposite side, acceptance tests, let the development team to discover different hidden requirements such as: functionality that has not been considered at the beginning of the project, requirements that were derived from the analysis of other requirements and requirements that were defined with a low detailed level. Finally, integration tests could be used to identify some hidden derived requirements. The next section describes a first step to define a systematic approach, to the identification of hidden requirements and its discovery in Agile methodologies. Figure 1 shows how each testing technique is related with the types of identified problems.

A first step to define a systematic approach to the identification of hidden requirements and its discovery in Agile methodologies is based on three main artifacts involved: testing, sprint review process and product backlog. The identification sequence is as follows: the development team (or automated testing tools) executes the test; for each failed test, it is checked if the cause for the failure is derived from the implementation or not. If implementation is the cause, e.g. some bugs or misunderstanding of user stories, corrective actions are taken. Otherwise, this means that something wrong has been done in the specification of user stories or in the specification of tests, all the issues that were wrong are analyzed to find out the causes behind. As result, some new user stories, representing new or missed functionality are included. The identification of these hidden requirements could improve the productivity of the development team, because the time spend re-working some parts of the project in futures iterations is reduced and more functionality could be done in the same time with higher levels of quality. Finally, as failed tests let the development team to identify hidden requirements, it could be interesting to put more efforts at the beginning of the process on acceptance test, because, as it was explained above, facilitates the discovery of requirements. It is supposed that in last iterations, there will be very few hidden requirements and, therefore, more efforts can be done in unit testing in order to improve the quality of the source code.

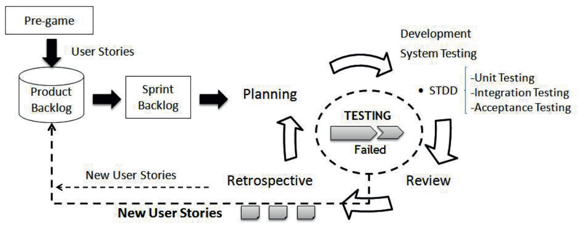


Fig. 2. Process Proposed to early identification of hidden requirements

4 Case of Study

As a first step to validate our approach, we have applied the process on an agile project for a product evolution. The project consisted in the evolution of a product called *TOPENprimer* to a biogas power plant and the name of the developed product is *TOPENbiogas*. The project was managed applying Scrum [23] as project management methodology and several XP practices in the development such as *pair programming* and *STDD*. The product was developed in 6 months by a 8 members team. More details about the evolution project are available in [7]. At the end of each sprint, failed tests in search of hidden requirements were analyzed. Some errors were produced because the expected behavior was not achieved. In the project, a simulator was used to support the development and some test failed in the simulator revealed that the real plant was working in a different way; therefore some biogas plant requirements were hidden. In other situations, the customer was focused in functional issues. When some acceptance test of a biogas closing gate were executed, a new hidden requirement was identified: after each error the system should stop the execution to prevent inconsistent states.

The analysis of using the process provided a couple of interesting conclusions. On the one hand, the early identification of hidden requirements could lead the process of building a solid software architecture from the beginning, decreasing rework and refactoring stages. On the other hand, it should be considered the time spent in review meetings. Review meetings were a little longer due to time devoted to analyze failed test, however, the development team consider that the time spent in these meetings is not comparable to the time that they should have spent on corrective action if the hidden needs had been discovered later. Nevertheless for the review of failed tests could be done by a part of the team.

5 Conclusions and Future Work

This paper has introduced two main issues: tests, failed tests, can be used to identify hidden requirements in agile processes. Therefore an optimized process can be implemented considering this feature. The second issue is that it should not be assumed that the role of process elements, e.g. tests, is the same in waterfall and agile. Actually, according to [22], tests can be considered as specifications. It might be necessary to fully rethink the process elements role and scope for agile. At present a formalization of the process to use failed tests as requirements enablers in Agile is being performed.

References

1. Boehm, B.: A view of 20th and 21st century software engineering. In: ICSE 2006: Proceedings of the 28th international conference on Software engineering (2006)
2. Cohn, M.: User Stories Applied: For Agile Software Development. The Addison-Wesley Signature Series. Addison-Wesley Professional, Reading (2004)
3. Grünbacher, P., Hofer, C.: Complementing xp with requirements negotiation. In: Proceedings 3rd Int. Conf. Extreme Programming and Agile Processes in Software Engineering, pp. 105–108. Springer, Heidelberg (2002)
4. Dyba, T., Dingsoyr, T.: Information and Software Technology, no. 9-10 (August 2008)

5. Moore, J., Abran, A., Bourque, P., Dupuis, R.: Guide to the Software Engineering Body of Knowledge 2004 Version. IEEE Press, Los Alamitos (2004)
6. Rodriguez, P., Alarcon, P., Garbajosa, J.: Identification of hidden requirements from failed system test execution. In: STV 2008 6th Workshop on System Testing and Validation - In conjunction with ServiceWave (2008)
7. Rodriguez, P., Yague, A., Alarcon, P., Garbajosa, J.: Agile methodologies from the perspective of the specification of functional and not functional requirements. In: 13th Conference on Software Engineering and Databases (JISBD 2008) (2008)
8. Woit, D.M.: Requirements interaction management in an extreme programming environment: a case study. In: ICSE 2005: Proceedings of the 27th international conference on Software engineering, pp. 489–494 (2005)
9. Robinson, W.N., Pawlowski, S.D., Volkov, V.: Requirements interaction management. ACM Comput. Surv. 35(2), 132–190 (2003)
10. Zowghi, D., Paryani, S.: Teaching requirements engineering through role playing: Lessons learnt. In: IEEE International Conference on Requirements Engineering, p. 233 (2003)
11. Eberlein, A., Leite, J.: Agile requirements definition: A view from requirements engineering. In: International Workshop on Time-Constrained Requirements Engineering, Essen, Germany (September)
12. Boness, K., Harrison, R.: Goal sketching: Towards agile requirements engineering. In: International Conference on Software Engineering Advances, 2007. ICSEA 2007, pp. 71–71 (August 2007)
13. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley Professional, Reading (2004)
14. Gallardo-Valencia, R., Olivera, V., Sim, S.: Are use cases beneficial for developers using agile requirements? In: Fifth International Workshop on Comparative Evaluation in Requirements Engineering, 2007. CERE 2007, pp. 11–22 (October 2007)
15. Ribeiro, J.C., Araujo, J.: Asporas: A requirements agile approach based on scenarios and aspects. In: Second International Conference on Research Challenges in Information Science, 2008. RCIS 2008, pp. 313–324 (June 2008)
16. Connolly, D., Keenan, F., Ryder, B.: Tag oriented agile requirements identification. In: ECBS 2008: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008) (2008)
17. Gupta, A., Jalote, P.: An experimental evaluation of the effectiveness and efficiency of the test driven development. In: First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007 (September 2007)
18. Desai, C., Janzen, D., Savage, K.: A survey of evidence for test-driven development in academia. SIGCSE Bull. 40(2), 97–101 (2008)
19. Ricca, F., Torchiano, M., Penta, M.D., Ceccato, M., Tonella, P.: Using acceptance tests as a support for clarifying requirements: A series of experiments. Information and Software Technology 51(2), 270–283 (2009)
20. Park, S.S., Maurer, F.: The benefits and challenges of executable acceptance testing. In: APOS 2008: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral, pp. 19–22 (2008)
21. Ho, C.-W., Johnson, M., Williams, L., Maximilien, E.: On agile performance requirements specification and testing. In: Agile Conference, 2006 (July 2006)
22. Mugridge, R.: Managing agile project requirements with storytest-driven development. IEEE Software 25(1), 68–75 (2008)
23. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River (2001)

Negotiating Contracts for Agile Projects: A Practical Perspective

Rashina Hoda, James Noble, and Stuart Marshall

School of Engineering and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand
{rashina,kjx,stuart}@ecs.vuw.ac.nz
http://ecs.victoria.ac.nz

Abstract. The Agile Manifesto values "customer collaboration over contract negotiation". However, in many real projects, Agile practitioners spend considerable time and effort negotiating contracts with customers. We have conducted grounded theory research in India with Agile practitioners. In this paper we present the strategies these practitioners use to overcome the problems of negotiating contracts. These strategies include changing the customers' mindset, providing different options of working, and — in the worst case scenario — keeping the customers unaware of internal Agile practices.

Keywords: Contracts, Agile Project Management, Grounded Theory.

1 Introduction

Agile practitioners often face challenges in adhering to their own agile principles. One such area where this occurs is the area of contract negotiation. During our qualitative research into the Indian Agile industry, we noted that most of our participants' customers demanded fixed bid contracts with fixed time, cost and scope variables. The practitioners explained that the customers felt this provided them with a perceived sense of predictability and control over the project schedule, cost, and deliverables. Since software development firms and their customers need legal contracts, this left the Agile practitioners to handle the apparent contradiction between the customers' desire for "certainty" with their own commitment to Agile values such as *responding to change* [1,2].

In this paper we report the results of qualitative research conducted in India. We identify the key challenges these practitioners face during contract negotiation, and present their proposed solutions to these challenges.

These results are a part of our larger research effort to explore the challenges and strategies of managing Agile projects using Grounded Theory [3]. In section 2 we will briefly describe grounded theory and present the parameters of our research and analysis. In section 3 we will present the results of our analysis, and then in section 4 we will cover related work. We will the conclude the paper in section 5.

2 Research Background

2.1 Grounded Theory

Grounded Theory (GT) is a qualitative research method developed by Glaser and Strauss [3]. GT is considered to be appropriate for research in areas that have not been studied before [4] and there is little academic research on the challenges of Agile project management in real world scenarios. The theory developed through this method does not have be to a universal truth, rather it needs to be a substantive theory describing processes in social organizations or communities [4]. GT researchers gather data and then systematically derive a substantive theory directly from that data, instead of first developing a theory and then systematically seeking evidence to verify it [4].

The researcher starts out with a general area of interest and gradually narrows their focus as they collect data from real life subjects. As we progress in our research, data collection, and analysis, we will move closer to developing a substantive theory. What we report in this paper are the major categories derived from the analysis of the data collected in India.

2.2 Participants and Procedure

We interviewed eight Agile practitioners from seven different software development organizations in India. The participants were using combinations of Scrum and XP . There were several Agile teams within the organizations. These teams used several Agile practices such as frequent releases, test driven-development (TDD), daily stand-ups, pair programming, release/iteration planning, continuous integration etc. The project duration varied from 2 to 4 months and the team sizes varied from 2 to 20 people on different projects. The products and services offered by the participants' organizations include web-based applications, front and back-office applications, and software development services. The interviewed participants were Scrum Masters and Agile coaches, except one who was a developer co-ordinating between the management and the rest of the team. The 8 practitioners included 2 CEOs of small scale firms who were certified Scrum Masters and had hands-on experience in working with their teams. We will keep the participants' identities confidential by refering to them only by number.

We conducted semi-structured, face-to-face interviews using open-ended questions. The interviews were recorded where permission was granted, and where the interviews were pre-scheduled. Then we started our coding [3]. GT coding involves the categorization, interpretation, and analysis of the collected data. We analyzed the data using the constant comparison method. This method requires that data from one interview or observation be compared to other pieces of data gathered from other interviews, observations and sources. Negotiating contracts for Agile projects emerged as a common category as a result of our data analysis, and we will now move on to discussing the results of this data analysis.

3 Data Analysis Results

Our participants mentioned contract negotiation as one of the main challenges they face in managing Agile projects.

> "sometimes limitations are imposed by customers, like technology or contracts...they just want to give you scope, requirements and expect you to deliver it or they are looking for a fixed price contract....if you ask me biggest problems...one is contracts...they want three things: fixed deadline, fixed price, and fixed scope." - Practitioner P3

Agile practitioners see fixed price contracts as a major limitation that the customers impose on them. Other practitioners shared their frustration over the issue of dealing with fixed time/scope/cost contracts, and their concerns on the impact that such contracts had on their ability to be agile and their ability to succeed.

> "Fixed price doesn't work well with Agile." - Practitioner P1

> "With Agile it's difficult to do fixed price projects. Agile takes about embracing change, can't do fixed price projects with changes coming in." - Practitioner P5

Our participants shared with us some of the strategies they used to deal with the customers' expectation of fixed bid contracts.

3.1 Changing Customers' Mindsets

> "All they [customers] have done is fixed price for last 20 years...very difficult to say it will not be fixed price." - Practitioner P5

Customers are used to fixed price/scope/time contracts. Our participants disclosed that it was difficult for their customers to change their ways of working to suit Agile projects. In a bid to resolve this issue of rigid mindsets, Agile practitioners often discuss the disadvantages of fixed bid contracts and the advantages of Agile development methods with customers. The same practitioner P5 shared the following property of Agile practices as an advantage to customers:

> "...focus is on delivering business value as soon as possible - as a result of that you take items which are most required from point of view of business, not the ones that are most interesting in terms of technical implementation." - Practitioner P5

Participant P8 noted that they often discuss with the customers how many features are seldom used. They also highlight how Agile allows the customer to avoid such situations by using prioritization of features, giving them more control of the product. Agile practitioners make an effort to change the mindset of the customers by encouraging them to look beyond the constraints of contracts, look at the bigger picture, and become convinced that Agile offers increased product control.

3.2 Providing Options

Agile practitioners offer different contract options to customers in order to encourage them to try Agile. Practitioners P3 and P8 encouraged customers to buy a few iterations to begin with instead of signing a contract for a large project up front:

> "Most of the time... [we] sell a certain number of iterations." - Practitioner P3

By allowing the customers to use Agile on a trial basis, Agile practitioners are able to build confidence among customers and provide them with risk coverage. Once the customers have tried a few iterations, then they are offered the option to buy more iterations or features as needed:

> "One thing we [development firm] used to do and worked very well - we used to tell the customers you don't have any risks...in case of Agile we enter into a contract with the client - OK we'll show you working software every fifteen days, you'll have the option of ending the project within one sprint's notice. Maximum they can lose is one sprint. Advantage we show to client they don't have to make up their entire mind [they] can include changes in sprints -they see it as a huge benefit to them." - Practitioner P5

> "Try for a month - then buy more sprints." - Practitioner P8

Some Agile practitioners allow the customers to swap features. The project is delivered at the same time and price as initially specified in the contract, but the customer can remove product features that they no longer require and replace them with new ones that are of more value to them.

> "...customer after seeing demo after 4th iteration realizes the features built, say the 13th feature, is not required and he needs something else...he can swap the two." - Practitioner P5

The practitioners also provide the customers with a termination clause in the contract such that customers have the option to quit on a few iterations' notice.

> "...[customers are] open to suggestions to retreat after few sprints." Practitioner P2

> "[Developers] start working on functionality from day one and you can add a sprint - not enter into contract for entire project - end in one sprint's notice and they [customers] can introduce change" - Practitioner P5

By providing the customers with the option to quit the project in the worst case scenario, some of their financial risks are covered. So if the customers are unhappy with the results, they can always quit the project.

3.3 The Last Resort

Some customers are still hard to convince so, Agile practitioners are forced to compromise with fixed bid contracts. In such situations, many Agile practitioners keep the customer unaware of the Agile practices being followed internally at the Agile organization.

> "the company had taken charge of the projects - we had made it Agile - internally following Agile, making frequent releases to customers and asking for feedback. So customer was not aware." - Practitioner P5

So while it seems like a traditional project to the customers, the development firm actually follows Agile internally at the team level. Sometimes Agile practitioners end up losing business as well.

> "...no match between what Agile says and the way they [customers] wanted. Yes, we lost business." - Practitioner P5

Ultimately, our participants' documented experiences confirmed that there are hard realities in practicing Agile methodologies in the real world. While these challenges can have serious repercussions on the participants' businesses (to the extent of losing a customer entirely), they continue to try to overcome these challenges with the different strategies discussed above.

4 Related Work

Many well known Agile practitioners and consultants have commented on the disadvantages of fixed price/scope/time contracts and have suggested their own solutions. Subramaniam and Hunt [5] suggest Agile practitioners should offer to build a small portion of the system on a trial basis. After the end of the iteration, the customer will have the option to continue or cancel the contract. This is reminiscent of the strategies of *providing options* discussed in section 4.2 that our participants have employed successfully.

Sutherland [6] introduces the concept of a 'change for free option' clause in standard fixed price contract. It allows customers to change feature priorities for free so long as the total contract work remains same. It also enables customers to "add new features if low priority items of equal work are removed from the contract." This is similar to the strategies used by our participants which allows customers to *swap features.* Franklin [7] discusses how they evolved from time and materials contracts to fixed price/scope/schedule contracts that supports Agile development at their organization. They conclude that developing a responsive contract modification process and building in buffer for schedule and scope changes are essential for success.

Our data analysis aligns with the work of these earlier practitioners and consultants, and can be viewed as further evidence supporting the argument that fixed bid contracts and Agile principles are not directly aligned, and that subsequently contract negotiation is a real issue for Agile practitioners.

5 Conclusion

We have conducted ground theory qualitative research in the India Agile industry. Our research has identifed that Indian Agile practitioners face a critical challenge in negotiating contracts and overcoming their customers' initial preference for fixed contracts. Our data analysis has uncovered some of the strategies employed by practitioners to overcome or mitigate this challenge. These strategies include changing mindsets of customers, providing different options of working, and in the worst case scenario - keeping the customers unaware of internal Agile practices. The strategies map on to similar ideas proposed by other researchers and practitioners in section 4, and our findings can be seen as supporting their arguments.

We plan to conduct follow-up interviews and observations with our practitioners. We will modify our future interview questions to focus on and explore these emerging categories.

References

1. Abbas, N., Gravell, A.M., Wills, G.B.: Historical Roots of Agile Methods: Where did Agile Thinking Come From? In: Proceedings of 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2008. LNBIP, vol. 9, pp. 94–103. Springer, Heidelberg (2008)
2. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to Agile methodologies. Commun. ACM 48(5), 72–78 (2005), `http://doi.acm.org/10.1145/1060710.1060712`
3. Strauss, A., Glaser, B.: The Discovery of Grounded Theory. Adline, Chicago (1967)
4. Adolph, S., Hall, W., Kruchten, P.: A methodological leg to stand on: lessons learned using grounded theory to study software development. In: Proceedings of the 2008 Conference of the Center For Advanced Studies on Collaborative Research: Meeting of Minds, pp. 166–178. ACM, Ontario (2008)
5. Subramaniam, V., Hunt, A.: Practices of an Agile Developer. Shroff Publishers, Mumbai (2006)
6. Sutherland, J.: Agile Contracts: Money for Nothing and Your Change for Free (2008), `http://jeffsutherland.com/scrum/2008/10/agile-contracts-money-for-nothing-and.html`
7. Franklin, T.: Adventures in Agile Contracting: Evolving from Time and Materials to Fixed Price, Fixed Scope Contracts. In: Agile 2008. IEEE Computer Society, Toronto (2008)

The Lego Lean Game

Danilo Sato and Francisco Trindade

ThoughtWorks Limited, UK
{DSato,FTrindad}@thoughtworks.com

Abstract. After revolutionizing the automobile industry, Lean principles have been applied to different knowledge areas, such as software development. However, many people haven't been introduced to the concepts that made Lean successful. In this interactive session, the participants will work in a small Lego^{TM} production line, experiencing the problems and applying Lean practices to overcome them. The workshop will also discuss the similarities and differences between the production line scenario and the software development industry.

Keywords: Lean Software Development, Kanban, Systems Thinking, Push System, Pull System, Work Cell.

1 Introduction

Toyota's approach to manufacturing and management philosophy has revolutionized the automobile industry, giving rise to what was later called *Lean* by those studying the success factors behind Toyota's approach [1]. Lean principles have been applied to different areas besides its origins in manufacturing, including: supply chain management, product development, and more recently software development [2].

Lean concepts are being proposed to solve software development problems, but most people trying to apply these solutions haven't had a chance to understand how these principles were originally applied in the manufacturing industry. In this 3 hours interactive session, participants will work in a small Lego^{TM} production line, experiencing the problems that arise in this scenario and understanding how Lean practices can be applied to overcome them.

Section 2 describes the expected structure and the workshop mechanics, while Section 3 discuss the expected outcomes for participants of this workshop.

2 Structure

The target audience of this workshop are practitioners with beginner or intermediate level of knowledge about Lean principles and practices. The number of participants is limited to 24 (minimum of 8) due to the nature of the exercises. They will be divided in 4 teams and will work in an imaginary production line to build Lego^{TM} houses.

Table 1. Workshop Mechanics

Description	Duration	Elapsed Time
Introduction	0:20	0:20
Iteration 1 – Hands on	0:15	0:35
Debrief/Retrospective 1 (Waste, Push vs. Pull, Kanban)	0:20	0:55
Iteration 2 – Hands on	0:15	1:10
Debrief/Retrospective 2 (Unlevelled Process, Systems Thinking, Work Cells)	0:20	1:30
Break	0:10	1:40
Iteration 3 – Hands on	0:15	1:55
Debrief/Retrospective 3 (Kaizen)	0:15	2:10
Group Activity (Improving the process)	0:10	2:20
Iteration 4 – Hands on	0:15	2:35
Lean In Software Development	0:20	2:55
Conclusion	0:05	3:00

The session mixes hands-on exercises with group discussions and slides to present the concepts incrementally. Each iteration is designed to demonstrate a different process to building the houses, as described in Table 1.

3 Expected Outcomes

– **Understand Lean concepts in a hands-on experience:** The participants will be involved in an interactive experience through the different aspects of Lean.
– **Demonstrate that Lean is more than just practices:** Approaching Lean concepts from their origin will allow participants to understand the core values, which can be further applied to different areas. With this understanding, attendees will be able to visualize why some Lean ideas can be very effective in software development, while others can not.
– **Have a good understanding of different Lean concepts:** Lean concepts like Waste, Push and Pull Systems, Just in Time, Systems Thinking, Work Cells, and Kaizen will be covered in the workshop, giving a broad overview of the main ideas behind the Lean movement.

The slides and the findings from the retrospectives will be published on the presenters' websites and will also be available for all conference attendees.

References

1. Liker, J.: The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer. McGraw Hill, New York (2004)
2. Poppendieck, M., Poppendieck, T.: Implementing Lean Software Development: From Concept to Cash. Addison-Wesley Professional, Reading (2006)

Agile Process Smells and Root Cause Analysis

Dave Nicolette

Abstract. A "smell" is an observable symptom of some underlying problem that can be perceived either by direct observation of the work flow or team dynamics or by examining trends in project metrics. In this workshop, we will explore common smells in agile processes, some suggested by the facilitator and some contributed by participants. We will use various root cause analysis tools and techniques to try and discover the underlying causes of the smells so that we can identify the appropriate corrective action in each case.

Keywords: Agile, process, methodology, management, root cause analysis.

1 Introduction

One of the fundamental ideas in agile software development is the notion of continuous improvement. Two of the twelve principles expressed in the Agile Manifesto [1] call attention to the importance of continuous improvement:

- Continuous attention to technical excellence and good design enhances agility.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile teams often find themselves caught up in the tactical demands of their projects to such an extent that it is difficult for them to pay continuous attention to excellence or to pause regularly to reflect on their effectiveness. The agile approach is often chosen for a project precisely because of challenging time-to-market needs or uncertainty about the detailed requirements for the software to be developed. Under these conditions, agile teams easily slip into a routine of following practices that may not be optimal.

Fortunately, there are observable symptoms of common problems that may be noticed by team members, project stakeholders, the project manager, the team coach, or others who have the opportunity to observe a team in action. Furthermore, simple root cause analysis techniques are available to help teams discover the underlying causes of the observable symptoms so that they can determine appropriate corrective action. This workshop concerns the symptoms to look for and the analysis techniques to use in following up on the observations.

2 Process Smells

The term *process smell* comes from the field of software development. It derives from the term *code smell.* A code smell is an observable characteristic of source code that suggests (to an observer who knows what to look for) that there may be deficiencies in the design of the code.

The term originated as a metaphor for smells in everyday life. If you noticed an unusual smell upon arriving home, you would naturally follow up to learn the source of the smell and determine whether it indicated an urgent problem such as a gas leak, a minor problem such as spoiled food in the kitchen, or no problem at all, such as the aroma of your neighbor's cooking wafting through the window.

Similarly, a process smell is a pattern of activity or a trend in metrics that calls for investigation to determine whether there is an urgent problem, a minor problem, or no problem at all in the team's work. In the workshop, participants will describe work patterns or trends in project metrics from their own projects that they perceive to be process smells. These will provide the inputs to root cause analysis exercises.

3 Root Cause Analysis

Having identified a process smell and recognizing a problem exists, it then falls to the team to understand the cause of the problem. In the workshop, participants apply a variety of root cause analysis techniques to the problems they bring from their own workplaces. These include at least the following:

- Five Whys
- Causal Factor Tree
- Diagram of Effects (Systems Thinking)
- Value Stream Map (Lean Manufacturing)
- Fault Tree Analysis, Ishikawa Diagram, Fishbone Diagram

As appropriate, some of the following techniques may also be applied:

- Failure Mode and Effects Analysis (FMEA)
- Barrier Analysis
- Change Analysis
- Apollo Root Cause Analysis (ARCA), Reality Charting
- Pareto Analysis
- RTR Problem Diagnosis (ITIL)

Reference

1. Agile Manifesto, `http://www.agilemanifesto.org`

Agile in Large-Scale Development Workshop: Coaching, Transitioning and Practicing

Thomas Nilsson and Andreas Larsson

Responsive Development Technologies AB, Teknikringen 10, S-583 30 Linköping, Sweden
{thomas.nilsson,andreas.larsson}@responsive.se

Abstract. Agile in large-scale and complex development presents its own set of problems, both how to practice, transition and coaching. This workshop aims at bringing persons interested in this topic together to share tools, techniques and insights. The workshop will follow the increasingly popular "lightning talk + open space" format.

1 Overview

Developing large-scale systems is both complex and difficult. Implementing Agile for large systems, or in large organizations, presents its own set of problems when practicing, transitioning and coaching. The organizers have worked and coached in both small and large-scale development for several years and would like to share. Without a doubt there are many experienced Agile coaches and practitioners who have similar, or completely different, experiences of working with Agile in large-scale development. Do you too, have something to share? Do you want to discuss a specific problem? Do you want to have a network of peers interested in this topic to exchange ideas, insights and questions? This workshop invites you to come together, connect and share.

2 Workshop Format

The workshop will consist of two parts:

- Lightning Talks – 5 minute presentations on a topic by participants
- Open Space – group discussions on proposed and selected topics

2.1 Lightning Talks

Lightning talks are short, intense, talks on a single subject, in this workshop limited to 5 minutes, after which the speaker must stop.

Participants are therefore requested to prepare a lightning talk on a topic relevant to the theme of the workshop and submit the title to the organizers at least a week before the workshop. In the workshop the participants will select 6 of the submitted talks that will be run as a warm-up and introduction to possible Open Space sessions.

2.2 Open Space

An Open Space session is a set of group talks on various topics held simultaneously. Participants can share, listen, and change groups at will. The participants will propose, prioritize and select topics within the scope of the theme for the workshop. Hopefully this is an opportunity to talk to, learn from and share with many other experienced coaches and practitioners that you might have a hard time finding during the rest of the conference.

Two rounds of Open Space sessions are planned including a short reconvening and summary from each group after each round.

3 Participants

If you are interested in Agile as applied to multiple teams, multiple products and/or complex systems of systems development, you are invited to participate. Agile coaches and practitioners, including developers, project and product leads, and managers, are included.

All participants are requested to prepare a 5 minute Lightning Talk on a relevant topic. The title of the talk should be sent to thomas.nilsson@responsive.se one week before the workshop at the latest. Participants are also encouraged to formulate suggestions for relevant Open Space topics. The suggestions need not be presented before the preparation of the second part on the workshop.

4 About the Organizers

Thomas has been coaching and teaching agile to individuals and organizations, small and large, for several years. He is the CTO of Responsive AB, a Swedish company specializing in enabling Agile transitions. Current projects include full time coaching a development site with around 100 software developers. Thomas has presented and hosted open space sessions on various conferences, *Agile in Sweden 2007*, *Scrum Gathering 2007* (London), *Agila Sverige 2008* (”Agile Sweden”) and *Agile 2008* (Toronto).

Andreas works as Senior Consultant and Agile Coach at Responsive. By actively taking a role as software developer, project leader, system integrator or scrum master in projects he has been doing, coaching and teaching agile to individuals and organizations since 2005. Andreas has presented and hosted open space sessions at several Swedish and international conferences, including *XP2006*, *Scrum Gathering 2007* and *Agila Sverige 2008*.

References

1. Gowans, S.: Rules of Open Space,
   `http://columbussocialmediacafe.org/rules-of-open-space/`
2. Wanted: Short Talks,
   `http://python.org/workshops/1997-10/shorties .html`
3. Fowler, M.: Giving Lightning Talks,
   `http://www.perl.com/pub/a/2004/07/30/lightningtalk.html`

What Does an Agile Coach Do?

Rachel Davies[1] and James Pullicino[2]

[1] Agile Experience Limited, United Kingdom
Rachel@agilexp.com
[2] British Broadcasting Corporation, United Kingdom
James.Pullicino@bbc.co.uk

Abstract. The surge in Agile adoption has created a demand for project managers rather than direct their teams. A sign of this trend is the ever-increasing number of people getting certified as scrum masters and agile leaders. Training courses that introduce agile practices are easy to find. But making the transition to coach is not as simple as understanding what agile practices are. Your challenge as an Agile Coach is to support your team in learning how to wield their new Agile tools in creating great software.

1 Workshop Summary

Richard Hackman claims in his book "Leading Teams" that there a three basic types of coaching intervention: Motivational, Consultative, Educational. We want to test out that theory and explore about what Agile Coaches really do.

The workshop explores typical situations that an agile coach faces. We aim to uncover specific coaching interventions that participants have tried. We also want to hear whether these interventions helped the team improve or not.

1.1 Participation

This workshop is for anyone who is introducing agile practices to teams. It will be suitable for most conference attendees whether they are managers, consultants, and educators. We assume participants are already familiar with at least one agile methodology and the most common agile practices.

To find out more about participation, please see our webpage at:
http://www.agilexp.com/XP2009-AgileCoachingWorkshop.php

2 Content and Process

Next participants will share their experience with the group by presenting their position papers. The workshop will then move into working in small groups to try an exercise in coaching agile practices. Following a break for coffee, we will debrief the exercise and discuss aspects of coaching revealed.

The workshop starts with introductions of both workshop participants and the topic. We will make a brief slide presentation on the role of an agile coach and Hackman's model of coaching interventions.

Next participants will move into working in small groups between 3-6 people around a table. We will distribute scenario cards (and also flipchart paper, sticky notes, and markers) between the groups. Each scenario card describes a typical problem an agile coach is likely to encounter. Here are a few examples: developers only make software available to testers on last day of iteration, daily standup meeting takes a long time, continuous integration tests are not fixed by team if they are failing, one developer objects to pair programming, managers put pressure on team to commit to more than their velocity, etc.

Each work groups starts by each person in the group writing down what interventions they would try as an agile coach faced with the scenario on the card. Each intervention is written on a separate sticky note and the notes from that group collected together. The work group then discusses the pros and cons of the different interventions. When each scenario has been discussed, it is passed to the next group (with the sticky notes written by the previous group). This is repeated so each group has worked on three or four scenarios.

The next step is to bring together the coaching interventions from all the groups into the following categories Motivational, Consultative, Educational, and Other. This should produce a flipchart sheet per category covered with interventions on sticky notes.

This opens into a discussion with the whole group on aspects of agile coaching revealed by the group work.

Now we start to distil the ideas into a mind map of what agile coaches do. Each work group will take a turn to present their mind map to the session group.

2.1 Timetable

The presenters will arrange to take digital photographs of all workshop outputs and arrange for these to be uploaded to the workshop web page or conference wiki website.

00:00 - 00:10	Introductions
00:10 - 00:30	Slide presentation
00:30 - 00:35	Get into groups
00:35 - 01:15	Scenarios/Interventions
01:15 - 01:30	Clustering
01:30 - 01:45	Break
01:45 - 02:15	Discussion
02:15 - 02:45	Mind maps
02:45 - 03:00	Review results

3 Workshop Organizers

Rachel Davies is a highly respected Agile Coach whose expertise is recognized internationally across the XP, Scrum and DSDM communities. She has 20 years experience in software development and started her own agile journey in 2000 as a programmer in an XP team. Rachel has served on the board of directors of Agile Alliance for 6 years. Rachel has presented at numerous conferences on topics related to agile coaching and has participated in the XP 200x conference program every year since 2001. She has also written a book on Agile Coaching which is to be published this summer.

James Pullicino works as a program manager at BBC. He has experience coaching agile teams and project managers within BBC. He has a special interest in Risk Management on Agile projects.

Workshop - Mechanics of Good - Creating Well Functioning Distributed Teams

Lars Arne Skår[1] and Jan-Erik Sandberg[2]

[1] Miles
[2] Det Norske Veritas

Abstract. What do those who succeed in having well functioning teams do differently from those who don't? After running workshops on different international conferences based on this subject it is clear that getting distributed projects right is still a challenge. Driven by the off-shoring trend, we see lot of great opportunities for improvement on team collaboration. The necessary compromises may have a significant influence of the outcome. The purpose of the workshop is to further explore what challenges exists and share experiences on what works and what doesn't; and why that is.

Keywords: culture, distributed teams, offshoring, organization, psychology, workshop, creativity.

1 Introduction

The content of the workshop will be based on the participants' contributions as well as the organizers. Furthermore, the outcome of a similar workshop conducted at Agile2008 in Toronto will be used as a basis for the workshop.

The organizers have been working in large software development companies who have been working with off shored developers in India, China and the Czech Republic, which has provided exposure and experience into what challenges exists. Probably the same kinds of mistakes have been done that almost everybody else has made until ways that works better was established.

The workshop will start with this initial list of potential topics to explore:

- The state of collaborative tools (last year it was still the simpler the better - the latest versions of the current tools still don't alleviate the distance problem for instance)
- Continuous build / integration across geographies site - how to manage, how to cope
- Onshore vs. offshore - how to divide work and coordinate (some best practices are emerging)
- Communication channels (why conference calls do not work - or how to make them work)
- How to justify the cost of travel - the need to meet face to face (might be challenging in these times - but was clearly an important finding last year)

- How to cope with cultural diversity, creating an atmosphere of mutual respect
- Buying into common practices across locations
- How to really know each other; how to bond across locations
- Experiences using a "proxy" to manage our offshore resources?
- First generation, second generation, third generation experiences in off shoring - how does it usually evolve/mature?

The topics coming from the participants will be prioritized, as those will be of the highest relevance.

We will be using a technique called "Goldtaking" to facilitate the workshop – see description in next chapter.

The group starts off by having a quick standup where everyone suggest one topic to discuss. These topics are being listed on a board/big notebook.

Everyone goes up to the board and make a single mark on one or two topics they like to discuss.

Depending on how many people are attending, the organizers choose a number of topics to discuss and make up groups for them based on number of markings. Since everyone can mark two topics, we easily filter the topics. If they could only select one, they are more likely to only mark the one they came up with in the beginning.

2 "Goldtaking" – New Workshop Technique

For each group we have a notepad and a pen and select a note taker to begin with. Whenever that note taker wants to make a statement or ask a question, she passes the notepad and pen clockwise, thus making the person sitting next to her the new note taker.

This has some interesting effects; one that we experience is that very often someone is likely to "take over" the discussion. After some time they are bound to become the new note taker, thus giving the word to the others.

Another thing that this accomplishes is that when we gather the books afterwards, we have an excellent document for making reports or post-documents of the discussion.

Everyone is free to take their own notes when they are not the official note taker, this means that if they really want to take notes on everything; they can either fill in their notes whenever they pass the notepad (after making the statement/question) or they can borrow the notebook afterwards and copy from it (the missing bits will be written by them self!).

The reason for the name "GoldTaking" is that this is inspired from Gold Fish Bowl-discussions and that the result document may be worth its weight in gold.

Test-Driven User Interfaces

Charlie Poole

NUnit Software, Poulsbo, WA, U.S.A.
charlie@nunit.com

Abstract. User interfaces have a reputation for being difficult to develop using TDD – and even to test in an isolated fashion. A number of techniques have demonstrated promise, but as new technologies arise, they often need to be re-invented or at least refurbished in order to remain useful. There is a strong need for common, cross-platform practices for isolated GUI testing and Test-Driven Development of GUIs.This workshop invites individuals with useful points of view, techniques and tools to present and discuss their ideas with other researchers and workers in the same field. Each participant is expected to prepare a brief presentation and to participate in active discussion of their own work and that of others.

1 Audience, Benefits and Outcomes

The workshop is primarily aimed at programmers with advanced TDD experience in one or more platforms, including GUI development and testing background, and at researchers in the same areas.

Participants will be able to share ideas with other individuals working in this rather sparsely documented area, hone those ideas and begin work toward a common standard of practice.

In addition to the individual benefits to the participants, this workshop will produce a number of recommendations for team and individual practices. Recommendations will be published on a web site or wiki – the specific location to be determined.

The workshop should – if successful – form a starting point for a long-lasting community effort to document practices of GUI testing and Test-Driven Development and to incorporate those insights in one or more open source tools.

2 Workshop Organization

The workshop is basically a micro-conference on GUI testing and TDD. It will be conducted in four phases.

1. In the initial phase, the facilitator will briefly summarize prior work in the problem space and ask the participants to set an agenda.
2. The second phase will consist of presentations by the workshop participants, each covering a specific approach, tool or technique for GUI testing or TDD. The presentations, having been submitted beforehand, will be grouped

according to general topics covered. Participants are expected to provide practical demonstrations of their ideas, illustrated by actual examples of tests or test methodologies.Presentations are time-boxed, as are the question and discussion periods following each presentation.
3. In the third phase, participants will be divided into separate small groups. Each group will be tasked to prepare a short declaration of lessons learned around a specific issue of GUI testing or TDD.
4. The final phase consists of the "lessons learned" presentations from each of the breakout groups.

3 Organizer

Charlie Poole has spent more than 30 years as a software developer, designer, project manager, trainer and coach. He is one of the authors of the NUnit testing framework for .NET and participates in a number of other agile tool projects.

Charlie has focused on GUI development on a number of platforms and languages. He provides training in GUI testing and TDD in the Windows environment, particularly under the .NET framework.

4 Prior Presentations

A tutorial on this topic was presented at XP2005 and various versions have also been presented commercially. This workshop is new in its present form and is intended to break new ground and identify new techniques for GUI testing and TDD.

The New New NEW! Product Development Game

Marc Evers[1] and Willem van den Ende[2]

[1] Piecemeal Growth, Delftseschans 18, 3432 TD Nieuwegein, The Netherlands
marc@qwan.it
[2] Living Software BV, Spilmanstraat 25, 5645 JE Eindhoven, The Netherlands
willem@qwan.it

Abstract. In this experiential workshop, we will explore the agile product development space (managing, planning, prioritizing, learning and scheduling) through simulating the different approaches, reflecting on our experiences, and summarizing what this means for our daily work.

Keywords: product development, agile planning, story mapping, dimensional planning, product backlogs.

1 Summary

There is more than one agile way to plan a project. Recently, a number of new product development methods such as Kanban, Dimensional Planning and User Story Mapping have sprung up to address shortcomings of 'traditional' agile planning methods in some contexts. Existing methods such as planning with ranges that are not yet mainstream are gaining tool support.

In this experiential workshop, we will explore the agile product development space (managing, planning, prioritizing, learning and scheduling) through simulating the different approaches, reflecting on our experiences, and summarizing what this means for our daily work.

2 Audience

Product owners, product managers, Scrum masters, project managers business analysts, researchers interested in product management, planning, and scheduling.

3 Process and Timetable

We will run short simulations using the different product management and planning approaches. We hand out printed stories for an example system to be used for planning. If the group is large, we will run two parallel simulations, otherwise we do it with one group. The groups get 45 minutes to plan the project using either:

- One of the existing variations of 'traditional' planning a la Scrum, XP or DSDM where stories are laid out in a linear fashion divided by iterations/sprints/timeboxes;
- One of the methods mentioned in the abstract;
- A method brought in by participants.

Time	Activity
30 min	introduction - short presentation of workshop goals and process; overview of different approaches
45 min	first simulation run
15 min	debrief, reflection
45 min	second simulation run
15 min	debrief
30 min	discussion - how to apply experiences in the real world

4 Presenters

Willem van den Ende is a Dutch eXtreme Programming pioneer. Since 1999 he guides organisations in being more effective, often through the introduction of Agile Software development as an all-hands person: coach, developer and facilitator. Always active in the local and international community, he serves as host of systemsthinking.net and the European Agile Open conferences and previously on the Agile Alliance board and co-founder of xp days benelux. Willem is an appreciated workshop facilitator at practitioners' conferences like XP(Day), Software Practice Advancement and Agile200*.

Marc Evers Evers works as an independent coach, trainer and consultant in the field of (agile) software development and softwareprocesses. Marc develops true learning organizations that focuson continuous reflection and improvement: apply, inspect, adapt. Marc also organizes workshops and conferences on agile and lean software development, extreme programming, systems thinking, theory of constraints, and effective communication. Marc is co-founder of the Agile Open and XP Days Benelux conferences, and founder and board member of AgileHolland.

Marc and Willem are partners in QWAN (Quality Without A Name – www.qwan.it), an initiative of pragmatic practitioners, who have joined forces to deliver courses and mentoring on effective software development.

Positioning Agility

Nilay Oza[1], Pekka Abrahamsson[2], and Kieran Conboy[3]

[1] VTT Technical Research Centre of Finland
nilay.oza@vtt.fi
[2] University of Helsinki, Finland
pekka.abrahamsson@cs.helsinki.fi
[3] National University of Ireland, Galway, Ireland
kieran.conboy@nuigalway.ie

Abstract. Agile methods are increasingly adopted by European companies. Academics too are conducting numerous studies on different tenets of agile methods. Companies often feel proud in marketing themselves as 'agile'. However, the true notion of 'being agile' seems to have been overlooked due to lack of positioning of oneself for agility. This raises a call for more research and interactions between academia and the industry. The proposed workshop refers to this call. It will be highly relevant to participants, interested in positioning their company's agility from organizational, group or project perspectives. The positioning of agility will help companies to better align their agile practices with stakeholder values. Results of the workshop will be shared across participants and they will also have opportunity to continue their work on agile positioning in their companies. At broader level, the work done in this workshop will contribute towards developing Agile Positioning System.

Keywords: Agility, Agile Software Development, Business Agility.

1 Description

The initiative of Agile positioning system stemmed from the FLEXI project – a pan-European ITEA project on Flexible Software Development techniques. More information about FLEXI is available at http://www.flexi-itea2.org As part of FLEXI project, we are working towards an Agile Positioning System (APS), a meta-framework that helps companies to better strategize agility from different angles and better address stakeholder values. APS is being developed with active cooperation from the FLEXI's industry and academic partners. An example of how an APS instance may look like is presented in Figure 1.

1.1 Expected Outcomes

Expected outcomes: The proposed workshop will generate instances of APS in different axes (shown in figure 1) and sub-instances of a particular axis. In particular, the workshop aims to gain diverse use of agile profiling from the participants as well as to use the stakeholder values as the key driver in identifying them.

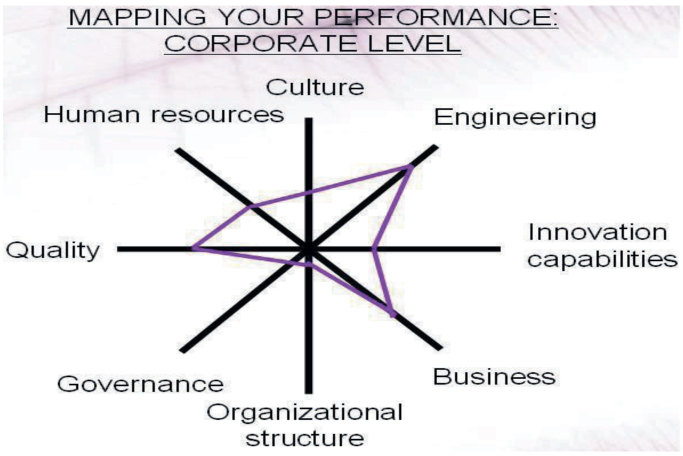


Fig. 1. APS instance

2 Workshop Outline

The workshop outline is as follows:

1. Introduction (10 minutes - personal introduction, participant expectations, workshop goals)
2. APS initiative and workshop flow (15 minutes)
3. Discussion on Higher level APS axis (10minutes)
4. Group work (30 minutes - each group with 3-5 participants, will work on one of the APS axis. Higher level APS axis will be proposed by organizers)
5. Break (10 minutes)
6. Group work continues (40 minutes)
7. Presenting the group work results (30 minutes)
8. Break (10 minutes)
9. Discussion on exploitation of results and next steps for APS (20 minutes)
10. Workshop feedback and closure (5 minutes)

The workshop will have participants working on APS. Interactions among break-out groups will be important to develop different instances of APS. Organizers will assist in maintaining workshop flow. The audience size should be between 25 and 35 with 3-5 participants in each break-out group. The main task for each group (for point 4 and 6 in above outline) will be:

1. Select one of the axes from APS
2. Discuss the axis in the group and develop an instance of APS for the chosen axis (similar to Figure 1). The group can develop sub-instances at-least upto two levels. More instructions will follow during the workshop.
3. Feed the axis data to a tool (will be provided in the workshop) to form a map similar to Figure 1. The tool for implementing APS data is under development in FLEXI project.
4. Develop acceptance criteria for the developed APS instance.

2.1 Benefits of Participation

Benefits of participation: The proposed workshop aims to develop other instances of APS with the participants. The industry participants will have opportunity to experiment APS in their own organizations and also be part of APS development. The academic participants will have opportunity to enrich theoretical angle of APS and identify new avenues of empirical studies.

3 About the Session Organizers

Nilay Oza is a senior research scientist at VTT, Technical Research Centre of Finland. He conducts research, develops and manages projects and offers advice to companies as a member of VTT. Nilay is VTT's leader in FLEXI project and recently he has been researching in the area of agile-innovation alignment, agile adoption, agile transformation, stakeholder values, distributed development, greenICT and lean manufacturing. Nilay can be reached at nilay.oza(at)vtt.fi

Pekka Abrahamsson is professor of computer science in the University of Helsinki in Finland. He is the project leader of the pan-European agile research initiatives and pursues to increase the agile capabilities in European companies and abroad. Pekka can be reached at pekka.abrahamsson(at)cs.helsinki.fi

Kieran Conboy is a lecturer in Information Systems at NUI Galway, Ireland. His research focuses on agile systems development approaches as well as agility across other disciplines. Kieran is currently involved in numerous national and international projects in this area, and has worked with many companies on their agile initiatives including Intel, Microsoft, Accenture, HP, and Fidelity Investments. Kieran has chaired related conferences including the European Conference in Information Systems (Galway 2008) the XP and Agile Development Conference (Limerick 2008) and also has chairing roles at XP2009 and XP2010. Some of his research has been published in various leading journals and conferences. Prior to joining NUI Galway, Kieran was a management consultant with Accenture, where he worked on a variety of projects across Europe and the US. Kieran can be reached at kieran.conboy(at)nuigalway.ie.

Scrum Board Game

Stefan van den Oord[1] and Wim van de Goor[2]

[1] SES – MiPlaza, Philips Research, Eindhoven, NL
stefan.van.den.oord@philips.com
[2] SES – MiPlaza, Philips Research, Eindhoven, NL
wim.van.de.goor@philips.com

Abstract. The Scrum Board Game is a workshop for beginners. It is for people with any role (customer, developer, tester, etc.), who don't exactly know what a Scrum Board is, or how to create one themselves. The workshop teaches the benefits of a Scrum Board, how to use it, and how to introduce it in projects.

Keywords: scrum board, game, workshop, pasta, communication, visibility, fun.

1 Workshop Overview

You are a team member on a pasta construction project. We will be your customer and we want to know how you're doing, if you will make the deadline, if you can take on extra work, etc. In this session, you experience how a scrum board can help answer these questions.

Using spaghetti, macaroni, and other pasta, you will construct furniture. We will help you discover how a Scrum Board can give your team and the customer insight in what's happening. You will learn that a Scrum Board gives visibility, clarity, improves quality, and is fun to use!

2 Workshop Goals

A typical project is not a straightforward exercise of performing tasks. As a customer, you may wonder what the developers are doing and how far they are. The development team may feel that the customer is micro-managing them, or is changing his mind every time they talk to him. They ask themselves how they can get the customer to understand the effect that has on productivity. Team members may have trouble answering simple questions like: "Can you make it this iteration?".

In order to answer these questions, you need a way to make the current situation visible. A Scrum Board is a very effective tool to make status and problems visible. This workshop is centered on learning how to create and manage a Scrum Board. Also, by actually using a Scrum Board during the workshop, teams learn to interpret their Scrum Boards. Not only that, but also how to come up with ways to make the desired information visible.

The goal of the workshop is for various stakeholders to learn the basics of using and interpreting a Scrum Board. Afterwards, you should be able to introduce a Scrum Board in your own project, and tailor it to your specific needs.

Last but not least: using a Scrum Board is fun, as demonstrated during the workshop.

3 Intended Audience

Participants should have no real experience with a Scrum Board. They may have heard about it, but they have no idea how to start using one in their project. It doesnt matter whether they are developers, testers, customers, project managers, etc. A mix of these roles is ideal, because that allows everybody to gain insight in the different ways to look at a Scrum Board.

4 Workshop Organizers

Stefan van den Oord became interested in XP in 2003. He introduced a number of XP practices in his company. In 2007, he joined SES (Philips MiPlaza), an organization with more than 8 years of experience with XP and Scrum. At Camp Scrum he learned the value of a Scrum Board. He now considers it an essential part of software development. His runs his own projects in an Agile manner, and coaches other teams.

Wim van de Goor is an Agile Software Team Leader at SES (Philips MiPlaza). He has more than 7 years of experience with XP and Scrum. He is eager to discover and try new practices. The Scrum Board was a successful practice that he started using in a complex project. Wim is also an Agile mentor and is involved in relentless improvement of the SES organization.

XP2009 Workshop: Climbing the Dreyfus Ladder of Agile Practices

Patrick Kua

Thoughworks
168-173 High Holborn,London
WC1V 7AA, United Kingdom
pkua@thoughtworks.com

Abstract. Adopting agile isn't as simple as ticking practices off a known list and moving onto the next one. It takes some effort to simply start applying a practice, and even more effort to attain a certain level of mastery. Distinguishing between apprentice-like behaviours and master-like behaviours is an important element of pushing the boundaries of a practice even more. This workshop uses the Dreyfus Model of Skill Acquisition to map behaviours we see in agile practices to the model and learn how to use this model to as a tool to help people progress to further levels of mastery within a particular practice.

Keywords: learning, Dreyfus Model, change, influence, organisational change, agile patterns.

1 Synopsis

Many learning models acknowledge people don't simply master a particular skill as soon as they start practicing with it, that the world is separated into those who have learned a skill, and those who are yet to learn it. These learning models acknowledge that there is often a scale between starting to use a practice, and using a practice effectively and to a point where they effectively apply a practice without any conscience effort.

Understanding how people learn is important for agile teams, both those starting to adopt practices, and those who are already applying them. For this ninety minute workshop, we will map behaviours we see of agile practices to the Dreyfus Model of Skill Acquisition[1] and spend some time understanding how to use it as an evaluation and coaching tool.

Specifically we will:

- Introduce how people learn and briefly mention a number of other learning models.
- Introduce the Dreyfus Model of Skill Acquisition
- Brainstorm behaviours that we see related to a finite set of agile practices

[1] Stuart E. Dreyfus; Hubert L. Dreyfus (Feb 1980), A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition.

- Classify those behaviours according to the levels defined by the Dreyfus Model
- Discuss how people might apply this map and model to their own situations
- Discuss how to keep the model up to date and relevant

2 Who Should Attend?

This workshop is aimed at anyone interested in how people learn, particularly about how people learn mastery of specific agile practices.

Whilst it will prove an invaluable tool for agile coaches, it should be useful for anyone currently applying agile practices or thinking about picking up agile practices in their own organisation. People should come away from their workshop able to recognise their "current state" and their desired "future state" with a clear and explicit model to share with others in their quest for mastery over several agile practices.

Ideal participants should have had some exposure to working with at least one agile team in order to relate to some of the behaviours and how to map them into the model.

3 Presenter's Background

Patrick Kua is an agile coach, facilitator and developer for ThoughtWorks. He has been working with individuals on teams in agile environments for the last five years, and understands how powerful and responsive people can be when working together in a common manner. He is always interested in aspects of continuous improvement, and how light weight processes can boost team effectiveness.

He brings a blend of deep technical skills and deep understanding of processes that help his teams succeed in their goals. He's presented at the last three XP200x about Test Driven Development, Information Radiators and Introducing Change into Organisations.

4 Workshop History

This workshop is based on a set of activities we've run for clients in the past, and thought it'd be useful as a way of sharing a wider variety of experiences in the agile community.

This will be the first time this specific workshop will be run at a conference.

The material in this workshop will be useful for anyone responsible for coaching teams in agile practices as a model for evaluating where people are in their mastery, and where they can go.

Software "Best" Practices: Agile Deconstructed

Steven Fraser

Director – Cisco Research Center, USA
sdfraser@acm.org

Abstract: This workshop will explore the intersection of agility and software development in a world of legacy code-bases and large teams. Organizations with hundreds of developers and code-bases exceeding a million or tens of millions of lines of code are seeking new ways to expedite development while retaining and attracting staff who desire to apply "agile" methods. This is a situation where specific agile practices may be embraced outside of their usual zone of applicability. Here is where practitioners must understand both what "best practices" already exist in the organization – and how they might be improved or modified by applying "agile" approaches.

1 Workshop Theme

What is "best" depends on context – and this workshop will explore software "best" practices in a hybrid world where the existing strategies for development invite agile to move beyond small co-located teams. Initial topics for discussion dependant on participant interests and experiences will include:

- What constitutes software "best practices" and their related limitations and failure modes – specifically in the context of large systems and teams.
- What constitutes team collaboration "best practices" and how they might be optimized in light of continuous change.

Where influences driving change might include:

- Acquisitions – integration of a team and its existing code base (often developed using agile methods, e.g. scrum and XP).
- New grads – integrating individuals who are generally more open to new practices and who don't fully understand the existing legacy code base.
- Increasing complexity – caused by aging software which is required to support continuous feature growth.
- The need for software archeology as teams are progressively restructured (continuous change in membership) and documentation loses its value.

2 Workshop Methodology and Anticipated Attendee Profile

The workshop will consist of a series of phases that foster a cycle of divergence and convergence to achieve shared understanding and foster networking and awareness by the participants. The brainstorming and working sessions will be facilitated using a variety of team techniques including Nominal Group Technique (NGT), Categorization, and Wideband Delphi facilitation methods. The proposed agenda is:

- First Part: Introductions and overview of best-practices.
- Second Part: Brainstorming and prioritizing ideas.
- Third Part: Structuring, summarizing and reporting of ideas.

The workshop is anticipated to attract managers, practitioners, consultants and academics who are interested with system development in an environment featuring large teams and legacy code bases – where the organization wishes to adapt agile methods to some, but perhaps not all, development practices.

3 Workshop Convener and Facilitator

STEVEN FRASER is the Director of the Cisco Research Center in San Jose California with responsibilities for developing and managing university research collaborations. Previously, Steven was a member of Qualcomm's Learning Center in San Diego, California with responsibilities for technical learning and development. Steven held a variety of technology management roles at BNR/Nortel including Process Architect, Senior Manager (Global External Research), and Design Process Advisor. In 1994, he was a Visiting Scientist at the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU) collaborating on the development of team-based domain analysis (software reuse) techniques. Fraser was the XP2006 General Chair, the Corporate Support Chair for OOPSLA'07 and OOPSLA'08, and Tutorial Chair for both XP2008 and ICSE 2009. Fraser holds a doctorate in EE from McGill University in Montréal - and is a member of the ACM and a senior member of the IEEE. Fraser is a trained (BNR/SEI) and experienced facilitator.

XP Workshop on Agile Product Line Engineering

Yaser Ghanam[1], Kendra Cooper[2], Pekka Abrahamsson[3], and Frank Maurer[1]

[1] Department of Computer Science, University of Calgary
{yghanam,fmaurer}@ucalgary.ca
[2] Department of Computer Science, University of Texas at Dallas
kcooper@utdallas.edu
[3] University of Helsinki, Finland
pekka.abrahamsson@vtt.fi

Abstract. Software Product Line Engineering (SPLE) promises to lower the costs of developing individual applications as they heavily reuse existing artifacts. Besides decreasing costs, software reuse achieves faster development and higher quality. Traditionally, SPLE favors big design upfront and employs traditional, heavy weight processes. On the other hand, agile methods have been proposed to rapidly develop high quality software by focusing on producing working code while reducing upfront analysis and design. Combining both paradigms, although is challenging, can yield significant improvements.

Keywords: agile product line engineering.

1 Introduction

Background. Software Product Lines (SPLs) are sets of similar, yet not identical, systems developed by an organization based on a set of core assets [1]. SPLs have proved to be effective in lowering the cost of software development, reducing time-to-market, and enhancing product quality. Traditionally, SPLE favors big upfront design and employs traditional, heavy weight software engineering processes. A domain architecture is usually required before individual applications are engineered. On the other hand, agile development methods such as Extreme Programming (XP) have been proposed to rapidly develop high quality software by focusing on developing working code while reducing upfront design and process overhead. Also, anecdotal evidence shows that Scrum is keenly adopted by large software companies basing their product development on SPLs. Scrum bases itself on cross-functional, self-organizing teams working in tightly time-boxed development settings. It is interesting to note that although the goals of the two software paradigms have similarities (time-boxed, high quality, complex software), the solutions to realize the goals seem to conflict and little work is available in the literature to integrate them. Integrating agile software development and SPLE is indeed challenging, but has the potential to magnify enhancements in quality, cuts in cost and reductions in time-to-market.

Goal and Scope. The goal of this workshop is to bring together people who are using or want to use agile approaches in the development of SPLs. We plan to discuss the similar goals but different philosophies of agile and product line development techniques, and try to explore to what degree they can (or should) be integrated, and how this integration can happen. Beside topics that will be proposed by the audience, we will try to address the following points: the needs of stakeholders (current and future), documentation, processes and activities to target for agility (domain engineering, application engineering, variability management), organizational software reuse, technology driven vs. customer driven projects, tool support, development culture & roles (architects, developers, testers).

2 Participation

Participants who wish to present in the workshop, are expected to submit a 2- to 4- page contribution under one of the categories below. The submission should be original. It should not have been accepted at or submitted to another venue (journal, conference, workshop, symposium). Submissions should be 2 to 4 pages in length.

Research papers: Original research on combining agile methods and SPLE. This can be either finished work or research in progress with preliminary results.

Experience reports: Stories of successful or unsuccessful attempts to integrate agile methods and SPLE. This is a great chance for people from industry to contribute their experiences and learned lessons.

Position papers: Thoughts and views about combining the two practices, supported by preliminary theoretical or practical work on the topic. This can also include the author's insight on current and future trends, or suggestions of best practices empowered by previous experiences.

3 Organization

Style. The workshop will be a half-day workshop. Participants whose work is accepted for presentation will be allocated time to present their submission, and engage in a Q&A session. Afterwards, a roundtable or "fishbowl" discussion on topics of interest to the audience will take place. The discussion will allow participants to engage in the workshop and ensure their questions are addressed. A summary of the discussions will be posted on a wiki page after the conference. Moreover, selected contributions will be included in a special issue journal.

Organizers. The co-chairs for this workshop are: Yaser Ghanam, University of Calgary, Canada; Kendra Cooper, University of Texas at Dallas, U.S.A; Pekka Abrahamsson, University of Helsinki, Finland; Frank Maurer, University of Calgary, Canada.

Reference

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2002)

Test Driven Development: Performing Art

Emily Bache

IBS JavaSolutions, Mölndalsvägen 77, SE-400 22 Göteborg, Sweden
emily.bache@ibs.se

Abstract. The art of Test Driven Development (TDD) is a skill that needs to be learnt, and which needs time and practice to master. In this workshop a select number of conference participants with considerable skill and experience are invited to perform code katas [1]. The aim is for them to demonstrate excellence and the use of Test Driven Development, and result in some high quality code. This would be for the benefit of the many programmers attending the conference, who could come along and witness high quality code being written using TDD, and get a chance to ask questions and provide feedback.

Keywords: Test Driven Development, Coding Dojo, Code Kata.

1 Background

The idea of code kata was introduced by Dave Thomas in his blog [1], and further developed by Laurent Bossavit and Emmanuel Gaillot [2]. One of the things Laurent and Emmanuel suggested was that after practice, you could become so expert at doing a kata that you could perform it for your peers at your coding dojo as an example of your best solution to the problem. The solution should be best not only in the sense that the resulting code is of high quality and well tested, but that you create it using a strict TDD approach, and can explain your design decisions to the group as you are writing it. In this way the whole group benefits from the experience of the individuals performing, and the individuals benefit from peer feedback.

2 Workshop Mechanics

The half day session will comprise of perhaps 6 kata performances. The performers will be chosen in advance, and perform in pairs. The remaining workshop participants will be there to learn from and critique the kata performances. The workshop moderator will introduce, moderate discussions during, and invite feedback after each performance.

Each performance will last 10-30 minutes, and be followed by 10 minutes of feedback/discussion, during which the next performers will set up their laptop with the projector.

3 Performer Selection

A public call for participation will be presented on codingdojo.org [3]. Two weeks before the conference the workshop moderator will make the final selection and publish a programme for the workshop, also on codingdojo.org. This programme will include timings and information about which kata each person is planning to perform, and which programming language they plan to use.

4 Expected Workshop Outcomes

The basic aim of the workshop is to help all participants to improve their practice of TDD. The code which each performance results in will be published online. If the kata is not already included in the catalogue on codingdojo.org, it will be writen up there.

5 Presenter Bio

Emily Bache is an experienced software developer and Test Driven Development practitioner. She regularly moderates coding dojo meetings and is a frequent contributor to the codingdojo.org website. Emily often speaks and leads tutorials at international conferences, for example agile2008, Scandinavian Developers conference and XP2007.

References

[1] `http://codekata.pragprog.com/codekata/`
[2] workshop "The coders dojo" at XP 2005 (2005)
[3] `http://www.codingdojo.org`

Business Value Game

Artem Marchenko[1] and Vasco Duarte[2]

[1] Nokia, Visiokatu 3, FIN-33720 Tampere, Finland
artem.marchenko@nokia.com
[2] Nokia, Kaj Franckin Katu 1A 8, FIN-00560 Helsinki, Finland
vasco.duarte@nokia.com

Abstract. Agile teams want to deliver maximum business value. That's easy if the on-site Ccstomer assigns business value to each story. But how does the customer do that? How can you estimate business value? This workshop is run as a game, where teams have to make tough business decisions for their "organizations". Teams have to decide which orders to take and what to deliver first in order to earn more. The session gives the participants basic business value estimation techniques, but the main point is to make people live through the business situation and to help them feel the consequences of various choices.

Keywords: business value, game, workshop.

1 Introduction

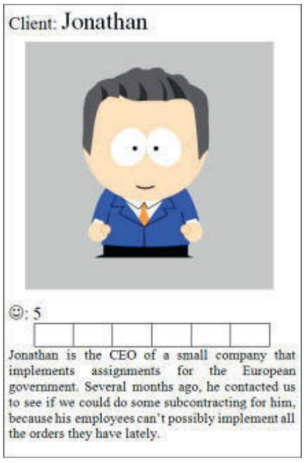
If you want to be a successful software development company, you have to make sure you work on the stories that bring most value to the customer. Ideally you have an on-site customer that can tell you which story can bring most value. What do you do if you don't have luxury of having the on-site customer? Or how does on-site customer decide what is of the biggest importance to him? How does your company prioritize epics, stories and projects? This session gives you some simple business value estimation techniques that are "good enough" for the everyday use.

2 Playing with Business Value

The session is run as a game, where teams of "businesspeople" have to make plans for their development team. The goal of the game is to earn as much money as possible by delivering features and stories with the highest possible business value, like in the XP Game. This game is a complement to the XP Game: how do these "business value points" on the XP Game story cards get chosen?

Each businessperson in the team represents one or more clients (Fig. 1) who will buy the team's product IF their feature(s) is in the product. The team is going to have to make some tough decisions; the team is going to have to disappoint some clients, because the development team has a finite capacity.

We provide the clients and their wishes. We suggest the techniques to estimate business value. The rest is up to you.


Client: Jonathan

☺: 5

Jonathan is the CEO of a small company that implements assignments for the European government. Several months ago, he contacted us to see if we could do some subcontracting for him, because his employees can't possibly implement all the orders they have lately.

Fig. 1. Client looking for more workforce

3 The Length of Workshop and Planned Number of Participants

The workshop takes 2-3 hours with a coffee break in the middle. When less time is available we cut out the non-critical steps, when more time is available we have an opportunity for larger and more productive discussions. When there are many people (e.g. 100), full three hours may be needed for the best results. The workshop can be played with any number of participants from 18 to 100. Ideal number is 30-60 people.

4 The Intended Audience and Expected Benefits of Attendance

Participants to planning games, especially product owners and on-site customers; business people; customers; those who teach or coach the preceding roles. No preparation is needed and the game should is of particular fun to the players new to agile.

Participants will learn how to assign business value to projects and stories, prioritize and make plans that bring value. They will also learn the ways for teaching this.

This game was originally designed by Vera Peeters and Pascal Van Cauwenberghe and was first tried on Agile 2008, then run on XP days Benelux and lately was a huge success on Scan-Agile conference in Helsinki.

Where Agile Research Meets Industry Needs: Starting from a 10-Year Retrospective

Xiaofeng Wang[1], Kieran Conboy[2], Minna Pikkarainen[3], and Michael Lane[4]

[1] Lero, The Irish Software Engineering Research Centre, Limerick, Ireland
xiaofeng.wang@lero.ie
[2] National University of Ireland Galway, Ireland
kieran.conboy@nuigalway.ie
[3] VTT Technical Research Centre of Finland, Oulu, Finland
minna.pikkarainen@vtt.fi
[4] University of Limerick, Limerick, Ireland
michael.lane@ul.ie

1 Summary

Research is often maligned for lacking relevance to industry. Does agile research meet industry needs? The workshop sets out to understand where they meet and identify the gaps between them. Starting from a close look at what have been investigated by agile researchers and what industry needs have been expressed by the time of the 10th edition of XP conference, we hope to draw a road map of high industry relevance for future research to exploit.

2 Audience and Benefits

The workshop targets at both researchers and people from industry. The active participation from both camps is crucial for the workshop to be successful. Therefore, in addition to publicizing the workshop on the conference website, we will publicize it through various channels of the organizers' institutions, and to their academic and industrial contacts. The workshop will be two to three hours in length, and open to the participants of the conference. There are no requirements for participation.

We hope to provide a forum where agile researchers can exchange their experiences and opinions and find a sense of community, and where agile practitioners express issues and concerns in their mind, therefore both inform and be informed by what has been achieved and what is going on in the research field. We plan to publish what we learn from the workshop as a research note or position paper.

3 Overview of the Process

The workshop will start with a pre-conducted survey of the research papers and experience reports that have been published in the XP proceedings from 2000 to 2009. Two taxonomies of agile topics will be constructed, which stand for the research topics explored and industry needs expressed respectively over the last 10 years. They serve as the start point of the workshop discussion which will be facilitated by the organizers and conducted as follows.

Step 1: Introduction to the workshop and preparation

- Introduce the objective of the workshop and how it will be conducted;
- Briefly present the two lists of topics.

Step 2: The audience will be divided into "R(esearch)" and "I(ndustry)" groups for group discussion. Each group will be facilitated by two of the organizers.

- For R group, there are three tasks to fulfill:
  (1) Extend the initial research topics list from their own research and research they are aware of. The extended list reflects the current state of agile research topics;
  (2) Further extend the list by adding the topics they wish to explore;
  (3) Prioritize the topics.
- For I group, there are two tasks to fulfill:
  (1) To extend the initial list of industry needs by adding their own needs;
  (2) Prioritize the topics.

Rationales behind each item on the lists and prioritization need to be elaborated.

Step 3: Representatives of the two groups report on the working results.
Step 4: Compare and combine the two lists to identify gaps between them and produce a unified list for future research.

4 Bios of Organizers

Xiaofeng Wang currently works as a research fellow in Lero, the Irish software engineering research centre. Her research areas include software development process, methods, agile software development, and complex adaptive systems theory. Her doctoral study investigated the application of complex adaptive systems theory in the research of agile software development. She has also worked in a major IT company in China and a research institute in Italy for several years in the area of enterprise knowledge systems. Her publications include several journal and conference papers in major IS journals and conferences.

Kieran Conboy is a lecturer in information systems at the National University of Ireland Galway. His research focuses on agile systems development approaches as well as agility across other disciplines. Kieran is currently involved in numerous national and international projects in this area, and has worked with many companies on their agile initiatives including Intel, Microsoft, Accenture, HP, and Fidelity Investments. Kieran has chaired related conferences including the European Conference in Information Systems (Galway 2008) the XP and Agile Development Conference (Limerick 2008) and also has chairing roles at XP2009 and XP2010. Some of his research has been published in various leading journals and conferences such as the

European Journal of Information Systems, the International Conference in Information Systems (ICIS), the European Conference in Information Systems (ECIS), IFIP 8.6 and the XP200n conference series. He is also associate editor of the European Journal of Information Systems. Prior to joining NUI Galway, Kieran was a management consultant with Accenture, where he worked on a variety of projects across Europe and the US. Kieran can be reached at kieran.conboynuigalway.ie.

Minna Pikkarainen has graduated from the Department of Information Processing Science, University of Oulu and finished her PhD about the topic of improving software development mediated with CMMI and agile practices at 2008. Minna has been working as researcher and project manager in VTT Technical Research Centre of Finland more than 11 years now. During that time she has worked in 18 industrial driven research projects doing close industrial collaboration with 8 organizations in Finland and in Ireland. Minna has participated as a key person for several large international ITEA project preparation work doing full project proposals and project outlines as collaboration together with large European level company networks (e.g. Flexi and Evolve projects). During 2007 – 2009 Minna has been leading VTT research group of the Large European projects called Agile ITEA (embedded agile software development) and Finnish consortium of ITEI (project about open innovations). Minna's research has been published in several journal and conference papers in the forums like ICSE, ICIS and Empirical Software Engineering Journal. So far Minna has provided agile trainings,workshops and invited talks for 10 different industries related to agile methods. Minna has been member of Lero, The Irish Software Engineering Research Centre since 2006. For the past 4 years, her work and publications have been focused on research in the area of agile software development.

Michael Lane is a lecturer in the department of computer science and information systems at the University of Limerick, Ireland. Michael's research interests revolve around the area of distributed software development. His PhD research is investigating project management in distributed teams leveraging agile software development practices. Prior to joining the University of Limerick in 2005, Michael had spent over 20 years in software development roles ranging from design and programming of bespoke services in the direct marketing sector to research and development manager of ERP products in the manufacturing sector. Michael's contact details are michael.lane@ul.ie.

Continuous Integration – How Do You Know That Your Application Still Works?

Thomas Sundberg

Agical AB, Stockholm, Sweden
thomas.sundberg@agical.com

Abstract. I will demonstrate how to develop a web application and have some degree of confidence that it still works after a developer has checked in new code or made changes to the existing code base. We will use Java as development language, Mercurial as version control system, Maven as build system, Hudson as Continuous Integration server, JBoss as application server, JUnit as primary test framework and Selenium to drive all GUI tests.

1 Intended Audience

Developers, managers and testers that develop web applications and want to find out how easy it can be to get started with automated testing and Continuous Integration.

We will use Java as development language, but the level of the programming will not be advanced. This is not a programming workshop. Any participant familiar with a c-like development language will be able to participate. C# developers should be able to follow the examples easily.

2 Learning Outcomes

The expected benefits for the participants are that they will not only have a gut feeling that automation is possible; they will know that it is doable and they have done it. They will have tools like Maven, Hudson and Selenium up and running so a project can be tested automatically every time a code base has changed.

Participants will know:

- How to set up a Continuous Integration environment with Hudson.
- How to integrate an automated testing environment for web applications using Selenium.
- How to build and test a project with Maven.

Participants that expect to get a deep knowledge about any particular tool will be disappointed. Developers that want to see that this mix of tools work together in a nice way will be able to see and verify it for themselves.

3 Process/Mechanics

This is a hands-on workshop. It will be divided into four parts.

The participants that want to take part in the programming need to bring a computer that is able to hook up to the network and on which they can edit some Java code. I will not require any particular development environment to be used. A simple editor and an installed JDK 1.5 or higher should suffice. Participants are expected to have knowledge and administrative rights enough be able to set an environment variable on there laptop and run a program from a prompt.

Part one
Introduction and setting up the development environment. I will have a few slides setting the stage and we will get Hudson, JBoss, Mercurial and Maven up and running on a build server.

Part two
Hands on installation experience that will show how easy it is to get a prepared environment up and running; I will supply the participants with a list of steps to perform for each developer and CD with all the required tools.

Development; I will supply each development pair with different tasks that can be implemented using test-driven development. The implementation can also be verified with a Selenium-driven integration test, run through a browser. I expect that it will be a lot easier if the participants work in pairs.

Part three
Setting up a local build server and trigger builds on it from the local Mercurial. That is to perform the same setup as I showed during the introduction.

Part four
Retrospective and discussions the last 15 minutes.

4 Presenter

Thomas Sundberg is a consultant at Agical AB in Stockholm, Sweden. He has a Masters degree in Computer Science from the Royal Institute of Technology, KTH, in Stockholm. He has been working as a Java developer the last ten years. His first experience with test driven development was with JUnit the autumn of 2000. He has also worked as a lecturer at KTH teaching programming courses. There he realized that students who solve programming assignments in pairs normally produce better solutions compared to students working alone.

Thomas has set up and maintained Continuous Integration systems since 2004 at different companies, including including a large Swedish-Japanese mobile phone manufacturer.

Executable Requirements in Practice

Pekka Klärck[1], Juha Rantanen[2], and Janne Härkönen[3]

[1] Eliga Oy, Finland
peke@eliga.fi
[2] Reaktor Innovations Oy, Finland
juha.rantanen@ri.fi
[3] Reaktor Innovations Oy, Finland
janne.harkonen@ri.fi

Abstract. Executable requirements neatly combine two important XP practices: *user stories* and *acceptance testing*. They enhance communication, ease following the number of *running tested features* during an iteration, and work as regression tests in future iterations. This workshop does not only give an introduction to this important process, but also shows how it is used in developing a real system in front of the audience. Some of the participants can even join the fun and get real hands-on experience.

1 Overview

Requirements in general and executable requirements in particular are important for all project members (customers, developers, testers, ...) and this workshop has something to offer to all these stakeholders. The workshop is suitable for both practitioners and beginners. Beginners will get a deep introduction of executable requirements and see how the process works. Practitioners can deepen their knowledge, share their experiences, and also get hands-on experience.

Workshop starts with short introduction of executable requirements and acceptance test driven development (ATDD) process. The ATTD workflow, *Discuss–Distill–Develop–Demo*, is explained to provide frame for rest of the session. Additionally Robot Framework[1], the tool that is used to automate acceptance test cases, is briefly introduced.

Preparations start by explaining an existing, partially complete application which will be developed further during a mini-iteration. Existing acceptance tests are demonstrated to provide understanding how acceptance testing has been conducted earlier. One participant is selected to play the role of the customer who will make the final decisions during the discuss and distill phases with help of the audience. Additionally ten *active participants* will be selected to be part of the development team and they will automate tests during the iteration in a coding dojo session. Organizers are leading the development team and are responsible for implementing the new features.

All the needed equipment is provided by the organizers.

2 Simulation

2.1 Discuss (10 Minutes)

Starting point of discussion is a prioritized list of user stories with development effort estimates. The customer selects items to be developed during the iteration and general discussion about the selected items follows.

2.2 Distill (20 Minutes)

High level acceptance test cases are written in co-operation between the customer and the development team. One of the organizers writes down the test cases in format supported by Robot Framework. Also other participants can take part in the discussion and help specifying the needed test cases.

2.3 Develop (90 Minutes)

Acceptance test cases are developed by the active participants with help of one of the organizers. Test cases are implemented in coding dojo style by changing the driver and co-driver for every five minutes. At the same time two organizers are developing features to the application using unit level test driven development. As soon as acceptance tests are ready, they are committed to the version control system. Progress is monitored using the automatically executed test cases.

2.4 Demo (15 Minutes)

Created functionality is demonstrated by running the acceptance test cases and manually exploring the new features. After the simulation there will be free discussion.

3 About the Session Organizers

Pekka Klärck (born Laukkanen) is a tester-developer and software contractor who works through his one-man-company Eliga Oy. Robot Framework concept is based on his Master's Thesis[2] he is still the lead developer of the framework.

Juha Rantanen and Janne Härkönen work as agile testing consultants at Reaktor Innovations, developing Robot Framework and helping customer teams in organizing their test automation efforts. Juha has also studied acceptance test driven development in his Master's Thesis[3].

References

1. Robot Framework project web site, `http://robotframework.org`
2. Laukkanen, P.: Data-Driven and Keyword-Driven Test Automation Frameworks (2006)
3. Rantanen, J.: Acceptance Test-Driven Development with Keyword-Driven Test Automation Framework in an Agile Software Project (2007)

Product Owners Jamboree

Patrick Steyaert and Tom Tourwé

patrick.steyaert@prosource.be
tom.tourwe@sirris.be

Brief Summary

Determining the features that should and those that should not be in your next product release is riddled with ambiguity, uncertainty and value conflict. Prioritising the product backlog according to business value is not easy when everyone has a different understanding of the meaning of value and everyone estimates the business value of a particular feature differently.

The product owner has a hard time reconciling management's strategic objectives with end-user's concerns; "incremental elaboration of the product" features with (disruptive) innovation features; features that bring money now with features that may bring money in the future; features that add functionality with features that add flexibility; market segment A features with market segment B features; etc.

Silver bullets - an enlightened product owner, business cases, multi-criteria analysis, etc. - may work in specific situations, but are hard to generalise or are not very sustainable.

In this workshop we will explore how wisdom of crowds techniques can be exploited to improve the definition of product releases. Can a group of individuals collaboratively improve its understanding of what "business value" means? Can the crowd provide more reliable value estimates? How can this acquired understanding of value be used to improve the selection of features for the next product release?

These are questions we strive to answer, by means of a hands-on session with Releasious, a tool that uses collaborative techniques to define product releases.

Expected Benefits for Participants

Participants will

- Learn how to turn the adagium "prioritise according to business value" into an operational definition.
- Learn about the "myths" surrounding product backlog prioritisation.
- Broaden their view on prioritisation and business value: this session is based on the newest state-of-the-art in the domain, resulting from the Flexi project. Flexi is the largest industry-oriented European research project on agile software development.
- Learn about lightweight methods, techniques and tools (developed during the same Flexi project) that they can try out and apply in their own context.

Description + Process/Mechanics

Our goal is to have a very interactive workshop, with minimal ex-cathedra presentations. Participants can use Releasious, our custom tool, that uses collaborative techniques to define product releases.

We will start by briefly presenting 5 "myths of product backlog prioritisation", as a motivation for our work. To introduce our state-of-the-art approach, we use a step-wise process: we will mix a presentation of the necessary concepts with interactive sessions, as described below.

During the workshop, participants will identify features for an example product and define their value in three steps.

- First, they will use the Releasious tool to individually provide example product features and define their value.
- Second, they will collaboratively define the "essential value drivers" for the example product (i.e. the criteria defining business value for the product) by means of a structured brainstorming workshop. Knowledge from the first step will be used as input.
- Third, they will use Releasious again to reconsider the value of their features, and provide feedback on the value of other participants' features. This time, they use the identified criteria and Kano analysis.

During the subsequent retrospective, we will analyse and discuss the results. We will compare the results obtained from the first step with those obtained from the third step, and compare the opinions of individuals with those of the group. This will also allow us to evaluate the proposed approach and tool, discuss advantages and disadvantages, and possible improvements.

An initial agenda looks as follows:

- Introduction
- Just-do-it: gather features for the example product and define their value
- Evaluation and Retrospective
- Think: define essential value-drivers for the example product
- Iterate: prioritise features using Kano analysis and value drivers
- Evaluation and Retrospective
- Workshop evaluation and retrospective

Organizers' Background

Tom Tourwé works for the Software Engineering group of Sirris, the collective center of the Belgian technology industry. Sirris advises and supports companies on the implementation of technological innovations, enabling them to strengthen their competitive position over the long-term. Tom is coordinating the Belgian activities in the Flexi project, the largest industry-driven European project on agile software development. While at Sirris, and during his previous academic career, he has organised over 20 workshops at many different events, conferences and symposia.

Patrick Steyaert is a Prosource Partner and Director of Assessments & Improvements. He is responsible for appraising the maturity of software processes of software organisations and for providing coaching on software process improvement programmes. He is an SEI trained CMMI specialist with more than 10 years of experience in software management of product as well as project development. He is a Certified ScrumMaster and has helped set up several agile software development teams.

Explaining the Obvious – How Do You Teach Agile?

Erik Lundh

Principal, Compelcon AB, Sweden
erik.lundh@compelcon.se

Abstract. Agile is now a hot topic and many organizations decide on adopting "agile" without really knowing how and why. This workshop will explore how fresh and seasoned agile coaches teach traditional and novel agile concepts, by example, with discussions. All participants are invited to show and tell about agile with an audience of peers. It might be the fresh first time with an audience, or golden hits that served you well for years.

Keywords: workshop, explaining agile, manifesting agile, canned experiences.

1 Introduction

We invite participants to share the proven and try the unproven with the audience. The short takes on how to teach and interact with managers, developers, product managers, project managers and of course were most of us started, with developers. How do we help people "get" everything from fundamental engineering practices to how to agile can fundamentally change a whole business for the better.

2 Workshop format

The organizer kick off the workshop with a few samples of "canned experiences" and explanation material distilled from 10 years of teaching and evangelizing agile to everything from .developers to top management, in everything from VC startups to large corporations like Ericsson and ABB.

Participants are then invited to give examples of the proven, and try new approaches in the preferred format of 2-5 minute lightning talks.

3 Workshop Organizer

ERIK LUNDH, Compelcon AB, has spent the last few years with Sweden's largest software development company group making a transition to agile for a R&D division with 2300 people on 10 sites in 5 countries, forming and working with an international group of 30 of the organizations own people as the agile coaches, supporting around 200 teams.

Erik has 10 years of experience of *successful* agile transitions and 25+ years in software development. He has served in most roles from developer to R&D manager and board member. Companies from small but mature innovation firms, startups to large organizations like Ericsson and ABB.

10 years as agile lobbyist and xp/agile coach in Sweden. Erik has attended all the XP200x conferences and has contributed since XP2001. Erik is a founder of the international Agile Coaches Guild, a sponsor of Swedens SPIN chapters, and an active industry member of the workgroup developing the first IEEE standard regarding agile, IEEE P1648 ("Recommended Practice for the Customer-Supplier Relationship in Agile Software Development") Agile Coaches Guild is an international network of people interested in the art and improvement of agile coaching.

Architecture-Centric Methods and Agile Approaches

Muhammad Ali Babar[1] and Pekka Abrahamsson[2]

[1] Lero, Univeristy of Limerick, Ireland
malibaba@lero.ie
[2] Univreisty of Helsinki
pekka.abrahamsson@cs.helsinki.fi

1 Overview

Agile software development approaches have had significant impact on industrial software development practices. Despite becoming widely popular, there is an increasing perplexity about the role and importance of a system's software architecture in agile approaches [1, 2]. Advocates of the vital role of architecture in achieving quality goals of large-scale-software-intensive-systems are skeptics of the scalability of any development approach that does not pay sufficient attention to architectural issues. However, the proponents of agile approaches usually perceive the upfront design and evaluation of architecture as being of less value to the customers of a system. According to them, for example, re-factoring can help fix most of the problems. Many experiences show that large-scale re-factoring often results in significant defects, which are very costly to address later in the development cycle. It is considered that re-factoring is worthwhile as long as the high-level design is good enough to limit the need for large-scale re-factoring [1, 3, 4].

There is a growing recognition of the importance of paying more attention to architectural aspects in agile approaches [4-7]. Researchers and practitioners have also identified the technical and organizational challenges involved in integrating Agile approaches in traditional software development methods [8, 9]. Hence, it is an emerging consensus among researchers and practitioners that there is a vital need for devising a research agenda for identifying and dealing with architecture-centric challenges in agile software development [10]. Such research agenda is expected to guide the future research on integrating architectural methods in agile approaches and give advice to the software industry on dealing with architecture related challenges.

2 Objectives and Format

The overall goal of this workshop is to refine and further develop the research agenda developed in the previous workshop. For developing such an agenda, the workshop participants will participate in several activities such as brainstorming, and discussion in order to reach consensus on important directions. To foster discussions at the workshop, the accepted papers will be put online before the event and the prospective participants will be invited to read them as preparation for the workshop. Moreover, we will also update the Wiki *(http://www.acube-community.org)* setup for the last workshop to reflect the research progress that has been made since the last workshop. During the previous workshop on this topic, held at XP08, the organizers and participants identified several research questions that are expected to stimulate the group discussions in the workshop. These questions are:

- How has the role of software architecture and software architects changed in projects using Agile approaches and its impact on the quality of the products and processes?
- What are the key architecture-centric challenges to implementing Agile processes for developing large scale systems and potential solutions available to practitioners?
- How can the considerations for non functional requirements be appropriately addressed while using Agile approaches?
- What are the prerequisites for integrating Architecture-Centric methods in agile development and potential implications of such integration on the processes and products?
- What kind of training needs to be provided in the areas of architecture-centric technologies (methods, techniques, and tools), customization and integration of architecture-centric approaches into Agile software development process?
- How can the use of patterns by agile developers help address architectural issues related to the required quality attributes?
- How does the architectural knowledge transfer from design to implementation in agile software development settings?

The workshop will be divided in two parts:

Part one will include short presentations of position statements and identification of controversial topics. Part two will consist of focused brainstorming and discussion activities in order to deliberate on the relevant issues and research questions.

References

[1] Kruchten, P.: Situated Agility. In: Proceedings of the 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering, Limerick, Ireland (2008)
[2] Kruchten, P.: Voyage in the Agile Memeplex. ACM Queue, 38–44 (July/August 2007)
[3] Boehm, B.: Get Ready for Agile Methods, with Care. IEEE Computer 35, 64–69 (2002)
[4] Ihme, T., Abrahamsson, P.: Agile Architecting: The Use of Architectural Patterns in Mobile Java Applications. International Journal of Agile Manufacturing 8, 1–16 (2005)
[5] Nord, R.L., Tomayko, J.E.: Software Architecture-Centric Methods and Agile Development. IEEE Software 23, 47–53 (2006)
[6] Parsons, R.: Architecture and Agile Methodologies - How to Get Along. In: WICSA (2008)
[7] Ali Babar, M., Abrahamsson, P.: Architecture-Centric Methods and Agile Approaches. In: Proccedings of the 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering, Limerick, Ireland (2008)
[8] Lycett, M., Macredie, R.D., Patel, C., Paul, R.J.: Migrating Agile Methods to Standardized Development Practice. IEEE Computer 36, 79–85 (2003)
[9] Boehm, B., Taylor, R.: Management challenges to implementing agile processes in traditional development organizations. IEEE Software 22, 30–39 (2005)
[10] Ali-Babar, M., Abrahamsson, P.: Architecture-Centric Methods and Agile Approaches. In: Proceedings of the 9th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2008) (2008)

3rd International Workshop on Designing Empirical Studies: Assessing the Effectiveness of Agile Methods (IWDES 2009)

Massimiliano Di Penta[1], Sandro Morasca[2], and Alberto Sillitti[3]

[1] Università degli Studi del Sannio, Italy
[2] Università degli Studi dell'Insubria, Italy
[3] Libera Università di Bolzano, Italy
dipenta@unisannio.it, sandro.morasca@uninsubria.it, alberto.sillitti@unibz.it

Abstract. Assessing the effectiveness of a development methodology is difficult and requires an extensive empirical investigation. Moreover, the design of such investigations is complex since they involve several stakeholders and their validity can be questioned if not replicated in similar and different contexts. Agilists are aware that data collection is important and the problem of designing and execute meaningful experiments is common. This workshop aims at creating a critical mass for the development of new and extensive investigations in the Agile world.

Keywords: Empirical investigation, data collection.

1 Introduction

Software engineering continuously introduce new development methods, techniques, and tools, which promise cost savings, reduced development time, better product quality, etc. However, when practitioners need to adopt a specific method, technique, or tool, they would like to have evidence that such method, technique, or tool produce the benefits promised.

Sensible evidence to substantiate claims can be provided via rigorous empirical investigation of methods, techniques, and tools in different contexts, to assess their strengths and weaknesses, and their suitability for specific needs of the practitioners. Such evaluations are complex since several variables need to be taken into account (e.g., background of the people involved, techniques used, working environment, technologies and tools adopted, etc.). Moreover, there are several different types of empirical study (e.g., surveys, case studies, and controlled experiments), each of which provides different degrees of strength and appropriateness to validate different kinds of empirical questions and contexts.

Rigorous procedures should be followed during an empirical study. When designing and conducting an empirical study, the investigator needs to make several decisions, including how to perform the study and which kind of data collect, e.g., measure the contextual factors that affect the study, choose the subjects and objects of study, etc.

These are a few possible problem that may arise if the investigator does not follow a rigorous procedure or makes wrong choices: data may fail to support even a true hypothesis, or, conversely, false hypotheses may be believed to be true because of inadequate evidence; insufficient documentation of contextual factors may prevent other people from adequately replicating a study, so the results may not be fully comparable and combinable; an insufficient number of data points may result in a non-statistically significant result.

In summary, the effort required to design and conduct an empirical study may be large and tedious, and many chances for error exist during the design of a study.

We believe that at least some of these problems may be addressed by brokering collaborations:

1. Allowing for replicating studies. This would produce several kinds of benefits, such as increasing the external validity of results (due for example to the heterogeneity of the employed subjects) or increasing their statistical power, by combining data from a series of experiments;
2. Collect and share "best practices" information regarding the design and conduct of empirical studies
3. Develop community-wide resources — e.g., analysis/recording tools, coding schemes, and repositories — that may be applied by many researchers to investigate many different research questions.

2 Goals

The goal of this workshop is to foster collaborations among researchers interested in conducting empirical research to assess the effectiveness of Agile Methods. The workshop is intended as a meeting forum where researchers will discuss and plan together joint empirical studies. As previously mentioned, the design of such studies is tricky, and there are many obstacles to replication.

A similar workshop has been organized at the International Conference on Program Comprehension (ICPC) in 2006 and in 2007. In such cases, the focus was on designing empirical studies for program comprehension. In this edition, we have decided to *focus on designing empirical studies for assessing the effectiveness of Agile Methods.*

In the previous editions, a number of open research issues have been identified. We hope to establish collaborations to reach a critical mass for the replication of a selected set of studies. Often, a single research group conducts a user study whose result suggests a trend but lacks statistical significance because the group cannot assemble a sufficiently large subject pool. If multiple research groups run independent studies using the same material, a meta-analysis may be able to combine these studies to yield a statistically significant result. By planning these independent studies and then performing a meta-analysis, we could learn much about how to deal with this general problem. To facilitate replication, one important issue to be discussed in the workshop is how to package and share materials and results to facilitate replication and make sure to learn from mistakes occurred in the first replications, e.g., improving the experiment design or material whenever needed.

Telling Your Stories: Why Stories Are Important for Your Team

Johanna Hunt[1] and Diana Larsen[2]

[1] University of Sussex, Falmer, Brighton, United Kingdom
j.m.hunt@sussex.ac.uk
[2] FutureWorks Consulting LLC, USA
dlarsen@futureworksconsulting.com
http://futureworksconsulting.com

Abstract. We all tell stories. The stories we share are shared to some purpose; e.g. to communicate ideals, to share knowledge, to warn, to entertain, to educate or show status. The story we tell changes depending on the context – when we tell it and who we are telling it to – and how we choose to tell it is also revealing of our values and underlying beliefs. This workshop is designed to explore the way we tell our stories, and practice telling and retelling stories through a series of collaborative story-games. Attendees will help to explore the possibilities for designing a set of storycards to help teams construct and tell stories around software projects.

Keywords: communication, culture, learning, organization, play, storytelling, team.

1 Workshop Overview

Stories surround us and shape our world. Stories can be the fairy and folk tales of our childhood, or just the mundane tale of "why I was late to that meeting." Storytelling is a medium of connectivity and community: the stories we choose to tell reflect our beliefs, our values and our fears and are a powerful tool for coordinating understanding and transferring knowledge and wisdom. Stories are interpretive, told by the teller to some purpose, enhanced, enriched and infused with meaning. They are thus inevitably more about the way an experience was understood, rather than 'the facts'. This is not a weakness of storytelling, but a chance to engage with the meaning as presented by the teller.

In recent years storytelling and narrative research has undergone a massive revival: psychologists study the way individuals understand themselves and form their identity through the stories they tell; social scientists look at the stories told by groups and communities; while the field of organizational storytelling (within business and organization studies) is daily gaining momentum into the study of how organizational culture, politics and change shapes, and is shaped by, the stories told by the individuals within it. While the implications and effectiveness of using stories as a tool for change is still under debate, the power of investigating and listening to them as a method of inquiry is undisputed.

This session aims to engage in a highly-participative fashion; by working with your own stories to look at values and cultures reflected by the stories told, to work with different interpretations of the same story, to consider collaborative storytelling for team-building, and to explore possibilities for designing a tool to help groups collaboratively tell stories together.

1.1 Content and Process

This session will begin with a brief introduction to stories and aspects of organizational storytelling and narrative research. We will warm up by collaboratively telling fictional stories using storycards. In groups, each person will then be asked to tell a story about a particularly memorable working experience they have had. The group will then choose a favourite story to retell and analyse further, before exploring possibilities and generating suggestions for a set of storycards to help teams tell and analyse their own stories. We will then draw out what people might find useful in considering stories in this manner, as well as their thoughts on how this could be used by teams for formation, co-ordination, and problem understanding.

1.2 Deliverables and Outcomes

By attending this workshop you will have the chance to learn more about the stories you tell – to explore different perspectives on the same story and consider some of the differing ways that stories can be value-laden – as well as participating in exploring possibilities for a set of storycards to help teams investigate their own values, beliefs and concerns through collaborative storytelling.

A summary of the workshop will appear on the Agile Narratives website (`http://www.agilenarratives.org`), a programme of the Agile Alliance.

2 Biography

Johanna Hunt is a PhD researcher looking at organizational storytelling during software development process change at the University of Sussex, UK. She leads postgraduate classes on research methods and group processes for interdisciplinary teams as well as acting as an agile coach and retrospective facilitator. Joh is co-chair of the Agile Alliance funded Agile Narratives project and has a passion for traditional storytelling.

Diana Larsen partners with clients in the software industry to support their efforts to create resilient workplaces, improve project performance, and thrive in times of change. In addition to consulting with teams and leaders on adopting agile approaches, she leads team, project, and whole system processes for collaborative thinking and planning. Currently serving as chair of the Agile Alliance board, Diana co-authored 'Agile Retrospectives: Making Good Teams Great!' with Esther Derby.

Elements of an Art - Agile Coaching

Erik Lundh

Principal, Compelcon AB, Sweden
erik.lundh@compelcon.se

Abstract. This tutorial gives you a lead on becoming or redefining yourself as an Agile Coach. Introduction to elements and dimensions of state-of-the-art Agile Coaching. How to position the agile coach to be effective in a larger setting. Making the agile transition – from a single team to thousands of people. How to support multiple teams as a coach. How to build a coaches network in your company. Challenges when the agile coach is a consultant and the organization is large.

Keywords: tutorial, agile coaching, scaling up agile.

1 Introduction

What is an Agile Coach? What is not agile coaching? How do I start and grow as an Agile Coach? Professional coaching organizations define coaching as helping people use the knowledge they already have. Agile coaches, however, often have to double as teachers bringing in knowledge to the team and the rest of the organization.

Some of us, proud to call ourselves seasoned agile coaches; never stand between the team and the people they have to work with. Most agile coaches cherish the moment when the team take over and fly!

What are the necessary skills for an agile coach? Of course we need to master how to do and coach key agile techniques like TDD, pair programming, continuous integration, refactoring etc. We need to help teams and their partners to be successful with low-fi, getting-things-done style planning and retrospectives.

But we also need to understand the different personalities (and their dynamics) of individuals and teams. How teams form and respond to changes. How different personalities interact. How different people learn.

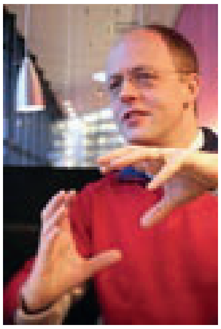
We will talk about having the courage to not stick with the team. Being a successful agile coach is often a transitional relationship with a team.

Different talents acting together as agile coaches can outperform the single hero coach. As an agile coach, we need to recognize our unique strengths, but not limit us to what we do best.

2 Format

We start with an introduction to elements and dimensions of state-of-the-art Agile Coaching. We then move to how to position the agile coach to be effective in a larger setting. Erik shares his take on making the agile transition – from a single team to thousands of people. And you get some insight in how to support multiple teams as a coach. If you are in a big organization you will learn something about how to build a coaches network in your company. We motivate this by discussing the challenges when the agile coach is a consultant and the organization is large. Erik also hopes to showcase some of the “canned” experiences and metaphors that are useful to explain agile.

3 Presenter



ERIK LUNDH, Compelcon AB, has spent the last few years with Sweden’s largest software development company group making a transition to agile for a R&D division with 2300 people on 10 sites in 5 countries, forming and working with an international group of 30 of the organizations own people as the agile coaches, supporting around 200 teams.

Erik has 10 years of experience of successful agile transitions and 25+ years in software development. He has served in most roles from developer to R&D manager and board member. Companies from small but mature innovation firms, startups to large organizations like Ericsson and ABB.

10 years as agile lobbyist and xp/agile coach in Sweden. Erik has attended all the XP200x conferences and has contributed since XP2001. Erik is a founder of the international Agile Coaches Guild, a sponsor of Swedens SPIN chapters, and an active industry member of the workgroup developing the first IEEE standard regarding agile, IEEE P1648 (“Recommended Practice for the Customer-Supplier Relationship in Agile Software Development”) Agile Coaches Guild is an international network of people interested in the art and improvement of agile coaching.

A Survey on Industrial Software Engineering

Adnan Causevic[1], Iva Krasteva[2], Rikard Land[1], Abdulkadir S.M. Sajeev[3], and Daniel Sundmark[1]

[1] Mälardalen University, Mälardalen Real-Time Research Centre, Västerås, Sweden
adnan.causevic@mdh.se
[2] Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria
[3] University of New England, School of Science and Technology, Armidale, Australia

Abstract. In this paper, we present on-going work on data collected by a questionnaire surveying process practices, preferences, and methods in industrial software engineering.

Keywords: Agile methods, Software Engineering, Testing, CBSE.

1 Introduction

Empirical research is vital for the success of the discipline of software engineering [1]. Well designed surveys are required to collect data for practical validation of hypotheses developed from theory and literature. This paper presents on-going work on data from a web-based questionnaire, surveying current processes, practices, and methods in the software industry. In the questionnaire, we studied three focus areas both separately and in combination:

- The adoption and benefit of different (particularly agile) **process practices**.
- The state of practice concerning **software components** in industry.
- The use, and sufficiency, of different **testing** methods.

The survey was anonymous, but respondents were given the choice to provide company and project name, to allow correlation of responses from the same organization. Respondent invitations were primarily sent to industrial partners in the FLEXI[1] and NESSI[2] european research projects. In addition, we encouraged the recipients to further spread the invitation.

2 Survey Contents

This survey included a comprehensive collection of questions based on process practices (e.g. [2]), component-based development and testing. The questions were divided into eight groups as shown in Table 1.

[1] www.flexi-itea2.org
[2] www.nessi-europe.com

Table 1. Question Groups in the Survey

Question group	**Purpose**
G1. Demographic aspects	Collect demographic data about the respondent.
G2. Project and product characteristics	Gather information about characteristics of the software and the project
G3. Software development process practices	Investigate software development process practices — both current and ideal situation.
G4. Software Testing	Collect information about software testing practices within the organization
G5. Component development	Collect information about component characteristics, as well as the development process.
G6. System characteristics	Gather system characteristics.
G7. System development	Collect information regarding component selection and system development.
G8. Discretional information	Optinal provision of organization and project name, to allow correlation of responses from same organization.

3 Ongoing Data Analysis and Future Work

We received 93 responses of which 42 responses were complete[3]. Since we encouraged recipients to further spread the invitation, we cannot determine the response frequency, nor which organizations are represented in the responses. This type of convenience sampling is suitable to collect empirical data exploring "how" questions, but during any statistical treatment of the data, we must consider the generalisability limitations imposed. Using both quantitative and qualitative analysis, we plan to study:

– Process practices and preferences, and industry demographics correlations.
– Environmental and project characteristics that are important when adopting agile development to different domains and project settings.
– Agility (e.g. embracing change) using black-box non-in-house components.
– System and component verification when reusing components in new contexts.
– Possible agile transitioning limitations by current testing techniques and tools.
– How traditional testers fit into an agile context and what their role would be.

References

1. Basili, V.R.: The role of experimentation in software engineering: past, current and future, Keynote presentation. In: 18th International Conference on Software Engineering, Berlin, Germany (1996)
2. Beck, K., et al.: Manifesto for Agile Software Development, `agilemanifesto.org`
3. Causevic, A., Krasteva, I., Land, R., Sajeev, A.S.M., Sundmark, D.: An Industrial Survey on Software Process Practices, Preferences and Methods, MRTC Report, Mälardalen University, Sweden

[3] Full survey response data is provided in [3].

Modeling Spontaneous Pair Programming When New Developers Join a Team

Ilenia Fronza and Giancarlo Succi

University of Bozen-Bolzano, Via della Mostra 4,
39100 Bolzano, Italy
{Ilenia.Fronza,Giancarlo.Succi}@unibz.it

Abstract. We present a study on how Pair Programming (PP) facilitates the introduction of new developers (novices) in a team. We analyzed the behavior of an industrial team of developers for 10 months focusing on spontaneous PP. During such time novices joined the team. Data has been collected non-invasively on how people paired during such time. Plots and sociograms are used to analyse such data and infer possible conclusions. It appears that initially PP is used to initiate the novices, then it is drastically reduced to be resumed eventually when the novices "feel" they have reached a significant level of maturity in the team.

Keywords: Pair Programming, Technology Transfer, Novices Integration, Team Social Network, Team Development, Sociograms.

1 Introduction

Pair Programming (PP) is claimed as an effective means for knowledge transfer when new members join a team [1],[6],[7]. The occurrence of spontaneous PP in a team is explored in [4]: novices are shown to do PP more during their first month in the team than veterans. In the second month, novices drastically reduce their time spent working in pairs, while experts do PP for about the same percentage of their total time. The analysis of the results in [4] raises two different possible scenarios:

1. Novices are completed integrated in the team after their first month of training.
2. The observed behavior is only a phase in a more complex process of integration.

2 Description of the Study and Results

We analyzed 10 months of work of a team, composed of 17 developers, 15 existing team members and 2 new team members, which joined the team at the starting of this research. The team is part of a software division of a large company and uses spontaneous PP and other XP practices. We focused on observing and trying to understand the effects of PP on novices integration. In this sense, this work is the continuation of the analysis started in [4], which analyses only a limited timespan.

We proceeded as follows:

1. We evaluated the monthly effort spent doing PP by the team as a whole and by the experts and the novices separately.
2. We built for each month a table containing for each developer who did PP the percentages of time (respect to his/her total work time) he/she spent with each of his/her partners.
3. We analyzed the trends through plots over time of the data found in the first step.
4. We used the tables built in the third step to create sociograms. Each developer is represented by a circle. The color of the circles represents the expertize of the developers and the size of the circles depends on the amount of time the developers spent in PP. Each circle is connected by edges to all the circles representing the developers he/she did PP with in that month. Edges thickness depends on the effort the two developers spent together.

The analysis suggests that PP promotes novices integration with a sequence of 4 well identified phases:

1. In the "initiation" phase (first month) novices spend a significant amount of time working with experts to be initiated to the new working environment.
2. During the second phase, novices aim at achieving "independence;" for about two months novices work mostly alone, consolidating their knowledge. When PP occurs, though, it is with experts.
3. During the third phase novices gain a "*maturity*" status in the team; they start back doing PP but mostly among themselves. This phase lasts about 4 months.
4. The fourth phase is the complete "*integration*" of novices in the team. PP occurs with all the team members in a way not different from the other team members.

The strengths of this approach are that:

1. It uses data from experienced developers working in their environment [3] instead of describing students behavior or professional working on controlled tasks.
2. It analyzes spontaneous PP, meaning that developers work in pairs spontaneously, that is, when they think that it is most effective.
3. The timeframe is quite large, 10 months, to follow the overall integration process of novices.
4. Sociograms are used to support the creation of the integration model.
5. We use the results of the observational study to suggest an integration model, which we compare with available group development models [2],[5].

References

1. Benedicenti, L., Paranjape, R.: Using Extreme Programming for Knowledge Transfer. In: Proceedings of 2nd International Conference on Agile Processes and eXtreme Programming in Software Engineering – XP 2001, pp. 75–78 (2001)
2. Charrier, G.O.: Cog's ladder: a model of group growth. SAM Advanced Management Journal 37, 30–38 (1972)

3. Coman, I., Sillitti, A.: An Empirical Exploratory Study on Inferring Developers' Activities from Low-Level Data. In: Proceedings of 19th International Conference on Software Engineering and Knowledge Engineering – SEKE 2007, pp. 15–18 (2007)
4. Coman, I.D., Sillitti, A., Succi, G.: Investigating the Usefulness of Pair-Programming in a Mature Agile Team. In: Proceedings of 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering – XP 2008. LNBIP, vol. 9, pp. 127–136. Springer, Heidelberg (2008)
5. Jones, J.E., Pfeiffer, J.W.: The 1975 Annual handbook for group facilitators. Pfeiffer & Co. (1975)
6. McDowell, C., Werner, L., Bullock, H., Fernald, J.: The effects of pair-programming on performance in an introductory programming course. SIGCSE Bulletin 34, 38–42 (2002)
7. Vanhanen, J., Korpi, H.: Experiences of Using Pair Programming in an Agile Project. In: Proceedings of 40th Annual Hawaii international Conference on System Sciences – HICSS 2007, pp. 274–283 (2007)

Summary Reporting for a Linked Interaction Design-Scrum Approach: How Much Modeling Is Useful?

Frank Keenan, Namgyal Damdul, Sandra Kelly, and David Connolly

Dundalk Institute of Technology, Dundalk, Co. Louth, Ireland
{frank.keenan,namgyal.damdul,sandra.kelly,david.connolly}@dkit.ie

Abstract. Identifying the minimum beneficial modeling to support an agile development team is crucial. Often, story cards arranged on wall charts or spontaneously drawn diagrams provide sufficient detail to allow a team to understand an emerging problem. However, what is beneficial when a new stakeholder joins a team after development has commenced and needs to have project background and progress reported? This poster reports on the models produced by a process combining aspects of Interaction Design (ID) and Scrum for internet development in such a scenario.

Keywords: Scrum, Interaction Design, Persona, Modeling.

1 Introduction

Despite the reported popularity and success of Agile methods (AMs), problems have been documented in their use, particularly for internet development. Usually, internet applications are designed and developed to be used by a particular audience. Before beginning development it is important to clearly identify the main user groups and their anticipated outcomes so that subsequent effort can be more focused and better managed. However, GUI-intensive projects are generally considered a challenge for AMs with user interface design largely ignored. One particular problem, in contrast to ID, is that AMs typically ignore consideration of user experience before development begins instead emphasizing collaboration throughout the project to supply any information required about users.

This research combined aspects of ID and Scrum to enhance the development of internet-based software development projects. Initially, stakeholders discuss the problem to ensure they begin with a common understanding, summarized in simple picture form. *Personas* and *goals* are used to provide the basis for writing *user stories* for the *product backlog*. The objective here is to examine the benefit of these models developed in reporting the problem overview and progress to another stakeholder joining the project briefly. This idea of someone parachuting into an agile development setting mid-project is common. Examples include; a new member joins a team during development, a senior figure in the development organization wants to get an overview or assess progress, or a new person takes over the customer role. Essentially, a key stakeholder wants to be brought up to speed as quickly as possible.

1.1 Evaluation

A study was conducted with the computing students, divided into two similar groups, with one following Scrum and the other the ID-Scrum process. Both teams were required to address the same problem, a Timetabling system, within 3 months, on a part-time basis. About midway through the project each group was required to present an overview of the project, progress to date and plans for the next iteration. A post-graduate student with no previous knowledge about the project, team, and status of the project performed this role. Each group was given 15 minutes to explain the project and answer questions. In both cases the visual artifacts were on display. For Scrum, this included stories organized on a chart with sections indicating: *backlog (to do), current iteration and work completed.* For the combined approach a *problem overview, picture persona charts* and *goals* were also displayed. Predefined questions were asked to each group relating to *project background, target users, specific goals, process followed, how was the problem determined, project progress, problems faced, plan to progress* and *changes in the plan.* The authors observed both sessions.

The Scrum group was able to describe their progress by demonstrating software developed to date and wall charts. However, labored responses were offered when asked about the general problem background, user groups involved and planned tests. The group frequently reverted to using technical terminology. A particular difficulty that was observed was the lack of focal point for the discussion.

The ID-Scrum Group used the problem overview diagram to illustrate their understanding of the problem highlighting different parts presented in diagram. The group members seemed quite confident in their understanding of the problem. This appeared accurate as they could point and refer to different parts of the diagrams. Multiple users had been identified and the need for the different features could be clarified from the overview diagram. They had a clear agreed definition of the project goal. The problem overview had helped in identifying the potential target users, stakeholders and their roles. Thinking like the personas had helped identify stories, prioritize these and identify key tests conditions. At this stage, *all* stories had been developed for the primary persona However, the personas and stories were not explicitly linked.

This student exercise, although limited, has provided positive feedback indicating that this combination, generating simple models, is beneficial in explaining a general project overview and progress to someone who has parachuted into a live project.

Software Product Line Engineering Approach for Enhancing Agile Methodologies

Jabier Martinez[1], Jessica Diaz[2], Jennifer Perez[2], and Juan Garbajosa[2]

[1] Fundación European Software Institute (ESI-Tecnalia)
Parque Tecnológico 204 E-48170 Zamudio, Bizkaia, Spain
jabier.martinez@esi.es
[2] Technical University of Madrid (UPM)
SYST Research Group
E.U. Informática. Ctra. Valencia Km. 7. E-28031 Madrid
yesica.diaz@upm.es, jennifer.perez@eui.upm.es, jgs@eui.upm.es

Abstract. One of the main principles of Agile methodologies consists in the early and continuous delivery of valuable software by short time-framed iterations. After each iteration, a working product is delivered according to the requirements defined at the beginning of the iteration. Testing tools facilitate the task of checking if the system provides the expected behavior according to the specified requirements. However, since testing tools need to be adapted in order to test new working products in each iteration, a significant effort has to be invested. This work presents a Software Product Line Engineering (SPLE) approach that allows flexibility in the adaption of testing tools with the working products in an iterative way. A case study is also presented using PLUM (Product Line Unified Modeller) as the tool suite for SPL implementation and management.

1 Introduction

Opposed to conventional evolutionary development in which stepwise refinement leads to the final product through a number of iterations, in Agile methodologies [1,2,3,4], a working product should be obtained after each iteration. It is also important that this working product should be built according to the requirements defined at the beginning of the iteration so the iteration will be successful only if the working product fulfills all these requirements. Therefore, a working product is delivered at the end of each iteration by means of continuous integration (CI) [5]. CI includes practices such as compilation, executing automated tests and software deployment. Testing tools provide automated support for the testing process. The benefits of using these tools have been proven in conventional methodologies. However, in agile methodologies, testing tools should be capable to test new working products in each iteration in a flexible and rapid way. In this sense, a significant effort has to be invested.

The contribution of this work consists in a SPLE approach for the adaption of testing tools with the new working products delivered in each iteration. A core concept of SPLE is to take advantage of common features among the products of a product line through the systematic reuse of these commonalities [6].

A case study is described to present our contribution. Particularly, the problem of developing a home automation system following SCRUM [7] and Test Driven Development (TDD) [8] is presented. The SCRUM methodology implies an incremental development and testing based on short time-framed iterations, called sprints. Assuming that each sprint implies the development and integration of a new home device in the home automation system, the development of a generic gateway, bridging the gap between any home automation system and a test environment, is desirable.

For this reason, the gateway should provide mechanisms to be adapted in a flexible and a systematic way, avoiding the redesign of a gateway for each specific device or starting it from scratch. A domain analysis has been made focusing on the gateway variability regarding on several types of possible devices and their communication peculiarities. This study shows that a SPLE approach is not only suitable for this problem, but also provides other benefits concerned to SPL such as domain knowledge encapsulation or final product deployment. A solution is illustrated using an independent domain SPL tool suite called PLUM (Product Line Unified Modeller) [9] that supports all the proposed PL life-cycle using a Model-Driven approach. This tool suite is used to manage gateway variability for specific gateways derivation in a flexible way.

Acknowledgments

The work reported here has been partially sponsored by FLEXI FIT-340005-2007-37 (ITEA2 6022) project and by UPM under their Researcher Training program.

References

1. The agile manifesto, `www.agilemanifesto.org` (accessed February 2009)
2. Cockburn, A.: Agile Software Development: The Cooperative Game, 2nd edn. Addison-Wesley Professional, Reading (2006)
3. Highsmith, J., Cockburn, A.: Agile software development: the business of innovation. Computer 34(9), 120–127 (2001)
4. Abrahamsson, P.: Agile software development methods review and analysis. VTT Electronics, 112, Tech. Rep. (2002)
5. Duvall, P., Matyas, S., Glover, A.: Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley Professional, Reading (2007)
6. Pohl, K., Böckle, G., Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Germany (2005)
7. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Englewood Cliffs (2002)
8. Beck, K.: Test Driven Development: By Example. Addison-Wesley, Reading (2002)
9. Aldazabal, A., Erofeev, S.: Product line unified modeller (plum). In: Eclipse Summit Europe (2007)

FLEXI Project Management Survey*

Anna Rohunen[1], Lech Krzanik[1], Pasi Kuvaja[1], Jouni Similä[1], Pilar Rodriguez[1,2], Jarkko Hyysalo[1], and Tommi Linna[1]

[1] University of Oulu, Department of Information Processing Sciences, P.O.Box 3000, 90014 University of Oulu, Finland
[2] Technical University of Madrid (UPM), E.U. Informatica, Ctra. Valencia Km. 7, E-28031 Madrid, Spain
{Anna.Rohunen,Lech.Krzanik,Pasi.Kuvaja,Jouni.Simila, Pilar.Rodriguez,Jarkko.Hyysalo,Tommi.Linna}@oulu.fi

Abstract. FLEXI Project Management Survey (FLEXI PMS) has been established to gain detailed knowledge on how the software industry – in particular successful companies – manages agile software development. FLEXI PMS investigates the actual agile values, principles, practices and contexts. The survey is supported by a careful literature review and analysis of existing studies. Special attention is attached to large, multi-site, multi-company and distributed projects – the target area of FLEXI project. The survey is intended to provide solid data for further knowledge acquisition and project/company positioning with regard to feasible agile management practices.

Keywords: Agile project management, agile software development.

1 Objectives

FLEXI Project Management Survey (FLEXI PMS) aims to discover good practices for project management, with emphasis on multi-site, multi-company, distributed software projects run by agile principles. The immediate objective is to survey the current state of the art in agile project management and provide solid data for further knowledge acquisition and project/company positioning with regard to agile management values, principles, and feasible practices. Data gathered through the survey undergoes exhaustive statistical analysis and comparisons with other research material. FLEXI PMS is supported by tools which focus on efficiency and usability of the survey process. The questionnaire is constructed to contain most critical items emerging in the field of distributed agile software project management, to be filled in a few minutes.

2 Background Material and Questionnaire

A collection of previous results, e.g., [1], [4], [5], has been carefully reviewed to define the scope of existing studies and their relevance to the FLEXI profile. Quality of

* This work has been supported by ITEA2 project E06022 FLEXI, "Flexible global product development and integration: From idea to product in 6 months."

reference studies was evaluated with regard to possible sampling bias as a consequence of not using a randomized sample. Measures were taken to avoid the problem by applying samples based on the wide range of various FLEXI partners' characteristics.

The questionnaire structure can be mapped onto three fundamental dimensions. One of them introduces a distinction between conventional agile and the extensions corresponding to the FLEXI profile: multi-site, multi-company and distributed projects – the latter occasionally contradicting with the original agile assumptions. Another dimension differentiates between agile values, principles, practices and tools, and reflects a context-driven approach [3]. The final dimension divides the survey into two stages: the knowledge acquisition stage and the company/project positioning stage. The optimal length, time needed to complete, clarity, as well as suitability and attractiveness for various stakeholder roles are important questionnaire characteristics which contribute to the survey's usability.

3 Results

Several experimental versions of FLEXI PMS have been released, populated and evaluated. They used dedicated web based survey tools. The current version [2] intends to deliver useful feedback already in early stages of the survey process. The respondents are provided with a report including aggregated data regarding the two stages – knowledge acquisition and positioning – subject to an assumed data security policy. Both stages ultimately intend to relate a survey contribution to the ways how successful companies are managing their agile software development. An exhaustive statistical analysis is employed to investigate the data. The report is generated immediately after taking part in the survey so that FLEXI PMS respondents can exploit the results in their agile development activity without delay.

References

1. Ambler, S.W.: Agile Practices and Principles Survey (2008), http://www.ambysoft.com
2. FLEXI Project Management Survey (2009), http://www.flexi-itea2.org
3. Kruchten, P.: Situated agility: Context does matter, a lot. In: Keynote, XP 2008, Limerick (2008)
4. Rumpe, B., Schröder, A.: Quantitative Survey on Extreme Programming Projects. In: Proc. 3rd Intl. Conf. on eXtreme Programming and Agile Processes in Software Engineering, Alghero, Italy (2002)
5. Vijayasarathy, L., Turk, D.: Agile software development: A survey of early adopters. Journal of Information Technology Management 19 (2008)

Documentation by Example

Daniel Brolund

Agical AB, Västerlånggatan 79, Stockholm, Sweden
daniel.brolund@agical.com
http://agical.com

Abstract. Writing documentation can be fun and rewarding, but keeping up with an ever-changing system can take a toll on that joy. The documentation tends to get either expensive (duplication-intense), outdated or non-existing. This demonstration will present an open source tool that addresses these shortcomings by extending the BDD[1] approach to provide rich and human readable documents automatically from a JUnit[2] test suite. You'll learn how to include snippets, run-time data and more in your documents, all this with minimal effort and intrusion. This approach is suitable both for APIs and GUIs, as will be shown.

Keywords: documentation, behavior-driven development, test-driven development, Java, JUnit, Bumblebee, example, snippet.

1 Intelligent Information Extraction

Write once, extract many is the central mantra of this session and this tool.

- By pulling information from a tested and executing system in an intelligent way, keeping documentation up-to-date is simplified.
- By intelligent naming of classes, methods and variables the effect is amplified, since those names can be extracted and used in the documentation, a natural extension of BDD.
- By keeping related example code and documentation text together, cohesion is improved, especially compared to the word-processor approach.
- By automatic inclusion of runtime data such as executing variables values, screen-shots, logs and more, even transient data is at hands reach. This provides a great complement to Javadoc[3].
- By starting with the end user documentation, it is easy to know what parts needs to be extracted and what really needs to be implemented. This will also reveal glitches in the system that TDD[4] won't expose.

This lessens the effort of keeping the documentation up to date which helps reduce time to market, waste and stress. The documentation itself provides views into the system that increases quality, understandability and is a great way to put usability in the drivers seat.

2 The Tool in Short

The tool presented is **Bumblebee**[5]. It is an add-on to JUnit that renders a *human readable document* each time the JUnit test suite is run.

The nested structure of the *test nodes*, i.e. the *test-suite/test-case/test*, will reflect in the document structure as *document nodes*, i.e. *section/sub-section/sub-sub-section*. The *camel-cased*[6] names of the test nodes will be *de-camelcased* when presented in the document nodes. An example of this is would be a suite- or case class name like `TheInnerWorkingsOfTheParser` that will become *The inner workings of the parser* in the document, or a test-method `howToConfigureTheParser` that becomes *How to configure the parser*.

In addition to the headers, comments in the suite class, test case class or test method becomes the paragraphs of the section. The text can be formatted using a wiki syntax. By the integration of JRuby[7] to execute the comment strings, various tools are available to automatically include code snippets, runtime data and more into the document, while at the same time allowing for extending the toolbox.

3 Implementations

The tool has been used in production for a number of different purposes; end user documentation of open source projects, e.g. dtangler[8] and Bumblebee itself, integrated system surveillance and problem solving manual, and developer documentation of the inner workings of an advanced Java-PL/SQL integration.

4 Audience

The primary target group is practicing Java developers that need to keep system documentation up-to-date while not loosing pace in development. Learning outcomes:

– Make documentation a fun part of every iteration without getting exhausted.
– Automate some of the most tedious parts of the documentation process.
– Keep the documentation in sync with the system.
– Make the documentation a part of the continuous integration loop.

References

1. BDD, `http://en.wikipedia.org/wiki/Behavior_driven_development`
2. JUnit, `http://junit.org`
3. Javadoc, `http://java.sun.com/j2se/javadoc`
4. TDD, `http://en.wikipedia.org/wiki/Test_Driven_Development`
5. Bumblebee, `http://www.agical.com/bumblebee/bumblebee_doc.html`
6. CamelCase, `http://en.wikipedia.org/wiki/CamelCase`
7. JRuby, `http://en.wikipedia.org/wiki/JRuby`
8. dtangler, `http://www.dtangler.org`

Alaska Simulator - A Journey to Planning

Barbara Weber[1], Jakob Pinggera[1], Stefan Zugal[1], and Werner Wild[2]

[1] Quality Engineering Research Group, University of Innsbruck, Austria
{Barbara.Weber,Jakob.Pinggera,Stefan.Zugal}@uibk.ac.at
[2] Evolution Consulting, Innsbruck, Austria
werner.wild@evolution.at

Abstract. The Alaska Simulator is an interactive software tool developed at the University of Innsbruck which allows people to test, analyze and improve their own planning behavior. In addition, the Alaska Simulator can be used for studying research questions in the context of software project management and other related fields. Thereby, the Alaska Simulator uses a journey as a metaphor for planning a software project. In the context of software project management the simulator can be used to compare traditional rather plan-driven project management methods with more agile approaches. Instead of pre-planning everything in advance agile approaches spread planning activities throughout the project and provide mechanisms for effectively dealing with uncertainty. The biggest challenge thereby is to find the right balance between pre-planning activities and keeping options open. The Alaska Simulator allows to explore how much planning is needed under different circumstances.

1 Introduction

The Alaska Simulator has been developed to support the teaching of planning approaches and to facilitate the execution of controlled experiments investigating the strength and weaknesses of different planning approaches. Due to the many similarities between software planning and journey planning the Alaska Simulator uses a journey as a metaphor [1]. The used metaphor is not only helpful to explain agile and lean principles to people without significant experience in the software engineering field, but is also an attractive context to be engaged in, thus increasing the willingness of students to participate in experiments. In the following we describe participating roles in the form of personas and how they can interact with and benefit from the Alaska Simulator

- *Steve Student:* tests and analyzes his planning behavior with the simulator and explores how much planning is just enough under different circumstances
- *Rose Researcher:* investigates the strengths and weaknesses of different planning approaches using the simulator

[1] For a detailed description of the journey metaphor see the simulator's website: http://www.alaskasimulator.org as well as the simulator's documentation.

– *Isabel Instructor:* demonstrates the different agile and lean principles using a journey as a metaphor and explains the major differences between agile, plan-driven and chaotic planning

The major features of the simulator are as follows:

– **Plan and execute journeys:** The Alaska Simulator allows to plan and execute journeys in either an agile or a plan-driven way.
– **Log journeys:** Each step that is performed while planning and executing a journey is logged.
– **Replay journeys:** To enable manual analysis of planning behavior, journeys can be replayed step by step.
– **Design journey scenarios:** The Alaska Simulator allows researchers and instructors to design their own journey scenarios including locations, actions, events, constraints as well as the degree of uncertainty (for details see [1]).
– **Analyze journeys:** Instructors and researchers are supported in analyzing the journeys after a planning session has been conducted.

Figure 1 illustrates the architecture of the Alaska Simulator which has been implemented as an Eclipse Richt Client Platform (RCP) application. For the Alaska Simulator itself three plug-ins `org.alaskasimulator.core`, `org.alaskasimulator.ui` and `org.alaskasimulator.help` have been developed. Additional components are the Alaska Configurator as well as the Alaska Analyzer.

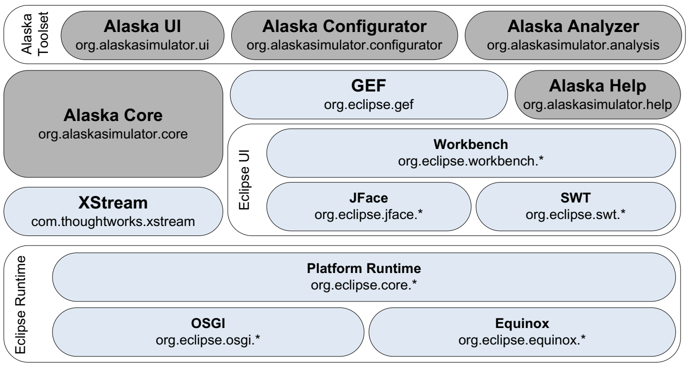


Fig. 1. The Alaska Toolset

The Alaska Simulator including a test configuration, extensive documentation and screencasts can be downloaded from http://www.alaskasimulator.org. The Alaska Configurator and the Alaska Analyzer are available to interested parties upon request.

Reference

1. Weber, B., Reijers, H., Zugal, S., Wild, W.: The declarative approach to business process execution: An empirical test". In: Proc. CAiSE 2009 (to appear, 2009)

Using Metric Visualization and Sharing Tool to Drive Agile-Related Practices

Tadas Remencius, Alberto Sillitti, and Giancarlo Succi

Free University of Bozen-Bolzano, BZ 39100, Italy
tremencius@unibz.it, asillitti@unibz.it, gsucci@unibz.it

Abstract. This paper presents a metric visualization and sharing tool that supports management and control of Agile-related practices, such as test-driven development, continuous integration, user stories, and pair programming. The tool is part of a larger framework but can be used as a stand-alone system. It integrates data coming from different sources: automatic non-invasive data collection plug-ins, bug and task tracking repositories, code parsers, manual user input, etc. The tool also provides customizable indicators that enable non-experts in the domain to get the general status of the observed process or product at-a-glance. The dashboard-based implementation of the tool is tailored to support multiple user roles, including developers, managers, and even clients.

Keywords: Agile, metrics, visualization, sharing, dashboard, indicators.

1 Introduction

Collecting measurements about development related processes helps to understand and control of what and why is actually happening. It is also a way of generating and preserving experience of the company. Unfortunately, a number of issues exist that have to be taken into account: the effort involved in data collection and metric analysis, user acceptance and support, making use of collected data, etc.

The tool we have developed aims to help address these issues. Instead of imposing new rules and procedures on the existing processes, it aims on reusing information sources already present in the company. This might be an issue tracking system, task database or simply an XML file generated by a test-run. Additional data are gathered by the non-invasive probes of the PROM framework [1], which the tool is part of. These probes collect mostly effort related data (e.g., time spent working on a specific class) in an automatic way with no visible impact on the performance of the monitored system.

The tool, called PROM Experience Manager (PEM) [2], is implemented as a web-based dashboard. It serves as a graphical interface for the employees to visualize, interpret and share software metrics. The idea is on one hand to provide each user with a customizable viewpoint on the collected measures, and on the other - to promote collaboration and experience sharing.

2 Supported Agile-Practices

The tool can help monitor and get a better understanding on different aspects of development process. The benefit it can provide, however, is largely dependent on each particular case. So far PEM has been used in one industrial and in two academic experiments. It showed that it was able to support four Agile-related practices: (1) test-driven development, (2) continuous integration, (3) user stories, and (4) pair programming.

PEM makes use of both unit test runs and test coverage information. Normally, this is extracted from XML files that are generated by development environments (such as Eclipse or Microsoft Visual Studio) or related tools and plug-ins. The tool has a server side scheduler that handles all data imports and caching. In a similar fashion the system gathers data related to continuous integration (e.g., from nightly-build logs of CruiseControl).

User stories and/or tasks are automatically imported from the repository used by the company (e.g., Bugzilla, Microsoft TeamSystem, etc.). Automatically collected effort is then mapped to user stories based on output of the Story Point Manager Utility (SPM), which is also part of the PROM framework. SPM is basically a manually controlled timer application that allows specifying the active user story and developers who are currently working on it (multiple in case of pair programming). Collected data are then integrated and visualized in PEM. Each user can create customized views (or widgets) and perspectives (collections of views) that can be shared with others. System-wide (visible to everyone) views and perspectives are also supported. Interpretation rules can be defined for each view and each perspective. These rules serve as mappings of metric values to an abstract scale and express the general meaning or status of the value of the metric. The concept that the tool is using is that of the "traffic lights": (a) red color stands for "bad", "problem", or "error" and implies that immediate attention is required; (b) yellow represents a "warning"; and (c) green color denotes that everything is "good" or "OK". When enabled, these interpretations are visualized as colored indicators and enable logged-in users to immediately see the current state of the monitored system at-a-glance. Furthermore, indicators allow non-experts to understand the meaning of metrics that normally require specific domain-knowledge. They also help to spot inconsistencies and conflicting interpretations of the same data by different users.

References

1. Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Collecting, integrating and analyzing software metrics and personal software process data. In: 29th Euromicro Conference, pp. 336–342 (2003)
2. Danovaro, E., Remencius, T., Sillitti, A., Succi, G.: PEM: experience management tool for software companies. In: 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, pp. 733–734. ACM, New York (2008)

ActiveStory Enhanced: Low-Fidelity Prototyping and Wizard of Oz Usability Testing Tool

Ali Hosseini-Khayat, Yaser Ghanam, Shelly Park, and Frank Maurer

Department of Computer Science, University of Calgary
2500 University Dr. NW, Calgary, AB
Canada, T2N 1N4
{hosseisa,yghanam,parksh,maurer}@cpsc.ucalgary.ca

Abstract. This paper presents "ActiveStory Enhanced" as a tool that enables prototyping user interfaces and conducting usability tests in a way that is aligned with agile principles. The tool allows designers to sketch user interface prototypes as well as add basic interactions to provide navigation. Sketching can be done using a mouse or stylus on tablet PCs. Designers can then export the prototype to a web-based Wizard of Oz testing tool, allowing test participants to remotely walk through a UI while recording metrics such as mouse movements and time spent on pages. ASE improves on the original by providing some usability improvements, improved browser support, undo support, more control over the design and an improved pen and paper metaphor.

Keywords: user-centered design, agile user experience design, tablet PC.

1 Introduction

With agile software development picking up increasing momentum and adoption over the past few years, some are concerned that a lack of attention to usability concerns has accompanied it[1]. Agile methods call for a reduction in the amount of time spent upfront on design details. User-centered design (UCD) on the other hand tends to involve a significant amount of upfront design work including prototype design and usability testing. Agile and UCD practitioners have now turned their attention to unifying usability engineering with agile practices [1, 2, 3, 4].

Low-fidelity prototyping involves drawing sketches of a prototype, usually on a piece of paper. This method has been shown to require less upfront work as well as focusing the participants' feedback on widget choice, widget placement and UI behavior in the early stages, rather than color choice, alignment and other less important superficial details [2]. Wizard of Oz testing consists of a person acting as the "wizard" while the test participant is shown the pieces of paper with the prototype designs, one page at a time. Whenever the user interacts with the interface, the wizard shows the next page which corresponds with that interaction. The drawback to this technique is that it traditionally requires the tester and participant to be collocated.

[1] Substantial traffic in the agile-usability newsgroup at Yahoo supports this statement.

2 ActiveStory Enhanced

ActiveStory Enhanced is a low-fidelity prototyping tool developed for the purpose of designing and performing distributed usability testing on an application. The tool allows designers to sketch user interfaces, add interactions and finally export the design to the web-based Wizard of Oz tool. Designers can use either a drawing tablet or tablet PC to further optimize the design experience and bring it closer to an actual pen and paper design feel.

2.1 Low-Fidelity Prototype Designer

The ActiveStory design tool allows designers to quickly draw user interfaces and link them together. The pen and paper metaphor is maintained with pen and eraser modes that allow the designer to draw on an ink surface. Interactions can be added to the design. In ActiveStory, an interaction is simply a region (hotspot) on a page that, once clicked upon, causes the system to load a new page. Designers can draw and erase with mouse or stylus; use custom pen colors, pen sizes and canvas sizes; select elements of the drawing and move, resize and delete them; add interactions and set the target page as well as resize and move the hotspot; and import existing images (e.g. screenshots or visual elements).

2.2 Wizard of Oz Testing Tool

The testing tool is Microsoft Silverlight based web application that presents the sketched prototypes to the test participant and handles navigations when a hotspot is clicked. It can be viewed by participants anywhere on the web, using any mainstream web browser and operating system. The Silverlight Wizard of Oz tool also collects some metrics that might be of interest to the usability designers and testers including: mouse behaviour (clicks and trails are presented to the tester by page, per user; or all the movements for all users on a given page are shown); time spent on each page (per user per instance of the page); clicks made on hotspots; and comments made by users on each page (per instance of the page).

References

1. Constantine, L.L.: Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. In: Information Age (August 2002)
2. Barnum, C.: Usability Testing and Research. Pearson Education, New York (2002)
3. Patton, J.: Hitting the Target: Adding Interaction Design to Agile Software Development. In: OOPSLA 2002, p. 1. ACM Press, New York (2002)
4. Sy, D.: Adapting Usability Investigations for Agile User-centered Design. Journal of Usability Studies 2(3), 112–132 (2002)

FitClipse: A Tool for Executable Acceptance Test Driven Development

Shahedul Huq Khandkar, Shelly Park, Yaser Ghanam, and Frank Maurer

Department of Computer Science, University of Calgary
2500 University Dr. NW, Calgary, AB
Canada T2N 1N4
{s.h.khandkar,sshpark,yghanam,fmaurer}@ucalgary.ca

Abstract. FitClipse is an Eclipse plug-in for facilitating Executable Acceptance Test Driven Development. The tool allows the users to edit acceptance tests, automatically generate fixtures, execute tests and represent the test results graphically including an option to view the test results history. The tool helps with regression testing because it can distinguish between requirements tasks that were never tackled before and tasks that were already completed but are now failing again. FitClipse currently supports GreenPepper and Fit framework.

Keywords: Executable Acceptance Test Driven Development, Testing Framework, Fit, Fitnesse, Greenpepper.

1 Introduction

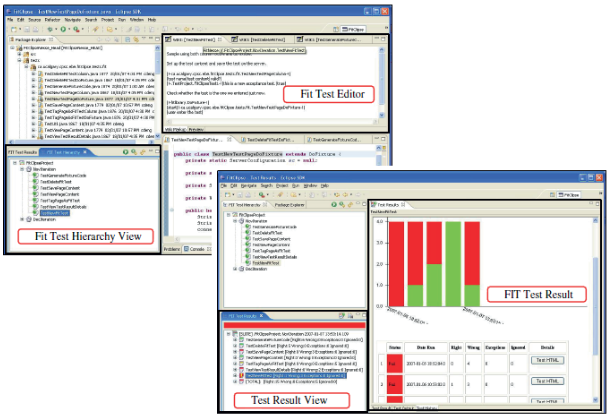
Extreme Programming ensures the quality of a software development project through unit testing and acceptance testing. Unit tests are used to verify the functionality of the software system from the developer's perspective and acceptance tests are used to verify whether the functionalities of the system meet the requirement of the customer. The agile methods suggest that there should be an acceptance test for every user story and a user story should not be considered completed until the software implementation passes all acceptance tests.

Executable Acceptance Test Driven Development (EATDD) is an extension of Test Driven Development (TDD). It is also known as Customer Test-Driven Development. While TDD focuses on ensuring the system design and stability from the developer's point of view using the unit tests, EATDD starts from the customer's perspective to help developers better understand the requirements and validate their development with the customer's requirements.

There are several commercial and open source frameworks that support automated acceptance tests. But due to the lack of proper IDE support, it becomes time consuming and tedious for developers to write and maintain these acceptance tests. FitClipse [1] is an Eclipse plug-in that solves this problem by providing the IDE support for common acceptance test frameworks and additional features to extend the benefits even more.

2 The Executable Acceptance Test Tool

FitClipse supports Fit [2] and Greenpepper [3] framework. The earlier version used a wiki based framework FitNesse [4] to store the tests. But many developers expressed that they want to store the tests with the development code to keep track of the changes better. However, FitNesse requires a web server to host the test specifications. To overcome this dependency, the current version of FitClipse replaced FitNesse with its own file system, which allows the code and the tests to be stored together in a version control system.



Fitclipse helps the test specification with a built-in WYSIWYG editor, which can help write the test specifications within the IDE more easily. Fitclipse can automatically generate the fixture stubs while you are designing the acceptance test specifications. Fitclipse comes with a test refactoring capability. It can synchronize any later change in the acceptance test (i.e. defining new test scenario, changes in test data) with related fixtures and helps reorganize the tests by automatically identifying the necessary fixture changes. FitClipse provides an analysis on the progress of the software development project using the test results history. The test result contains a statistical report and possible causes of test failures (i.e. exceptions). Often there is considerable delay between defining the acceptance test and its first successful pass. It is important to be able to distinguish between tasks that were never passed before and tasks that were already completed but whose tests are now failing for regression testing purpose. FitClipse is available for download at http://ase.cpsc.ucalgary.ca/ index.php/FitClipse/ FitClipseFrameWorkInstallation.

References

1. Deng,C., Wilson, P., Maurer, F., FitClipse: A Fit-based Eclipse Plug-in For Executable Acceptance Test Driven Development, Proc. of the XP 2007, Come, Italy 2007, Springer
2. Fit, http:// fit.c2.com
3. Greenpepper, http://www.greenpeppersoftware.com
4. Fitnesse, http://fitnesse.org

Using Digital Tabletops to Support Distributed Agile Planning Meetings

Xin Wang, Yaser Ghanam, Shelly Park, and Frank Maurer

Department of Computer Science, University of Calgary
2500 University Dr. NW, Calgary, AB
Canada, T2N 1N4
{xin,yghanam,parksh,maurer}@cpsc.ucalgary.ca

Abstract. Digital tabletop is an emerging technology that is being increasingly used to support group activities. Agile Planner for Digital Tabletops (APDT) is a tool that was built to support agile planning meetings. It provides interactions similar to those used in traditional pen-and-paper meetings. Previous versions of APDT were only capable of handling collocated planning meetings. In this paper, we succinctly describe the new extension of APDT that provides support for distributed planning meetings. A series of evaluations for the new version of APDT has been conducted, and the feedback tends to be positive.

Keywords: Agile Project Planning, Digital Tabletop, Group Collaboration.

1 Introduction

Planning meetings are an essential group activity in software teams, especially agile teams. They involve software developers, customers and other stakeholders. They are often held before the beginning of a new iteration. Agile planning meetings can be either collocated (stakeholders in the same physical space), or distributed. Previously, we have studied and provided insights on how to use digital tabletops (APDT) for collocated planning meetings [1]. When talking about distributed settings, however, the matter seems to be much more challenging. Distributed planning meetings might deteriorate the cohesiveness of the meeting flow, as well as the comprehension of what needs to be done.

Understanding the intricacies of distributed settings, here we discuss the new capability of APDT that enables distributed agile planning meetings. Usually, remote stakeholders find it hard to communication in a spatially separated environment. Therefore, some tools like DAP [2] were employed to help distributed communication. There are some obvious shortcomings. First, it is hard to keep all ends of communication synchronized. Second, there is no mechanism to encourage remote communication – in some cases remote communication might even be discouraged because of how unnatural meeting interactions become. APDT attempts to overcome these shortcomings and provide mitigation for the challenges surrounding distributed planning meetings.

2 Agile Planner for Digital Tabletop (APDT)

The new capabilities of APDT were designed to support activities of distributed project planning, and migrate natural behaviours from traditional settings to computer-based environments. APDT utilizes interaction features of digital tabletop to enhance group collaborations and bridge communication gaps among distributed teams. Figure 1 shows a live meeting between two groups located in two different physical spaces.

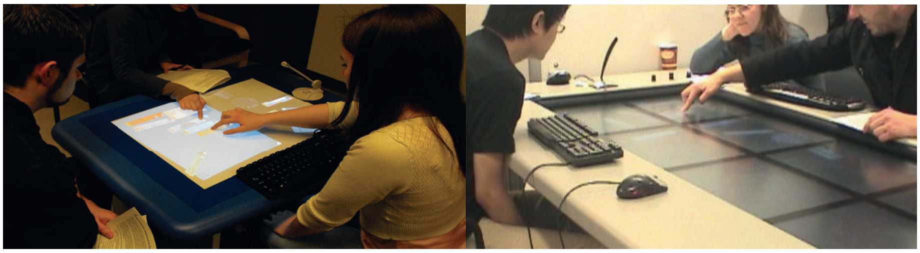


Fig. 1. Tabletop based distributed agile planning

APDT allows for multimodal interaction with the digital tabletops. It implements: 1) finger touch or mouse events 2) gesture recognition, 3) handwriting recognition, and 4) voice command recognition. To support distributed collaborations, telepointers (remote mouse pointers) are used so that the finger movement of the participant at one location could be broadcasted to every other location in the meeting scenario. Story card operations, such as creating/deleting cards are also supported. APDT can be connected with other agile planning platforms. A gateway to other team applications is developed so that the XML formatted data of project meeting from APDT is easily shared by other applications. At present, APDT can communicate with IBM Jazz [3] and Rally [4]. We have deployed APDT on SMART Board and SMART Table. SMART Board has a 183cm x 122cm screen, employing DViT technology to support at most 2 concurrent touches. SMART Table has 55.9cm x 41.9 cm active screen area, utilizing TFIR technology to support about 40 simultaneous touches. Although we still have some issues to tackle in our solution, our initial evaluation of the distributed aspect of APDT, in controlled settings, yielded positive results. We intend to extend the study to observe real settings, and report the outcomes in the near future.

References

1. Ghanam, Y., Wang, X., Maurer, F.: Utilizing Digital Tabletops in Collocated Agile Planning Meetings. In: Proceedings of the Agile Conference 2008, Toronto (2008)
2. Morgan, R., Walny, J., Kolenda, H., Ginez, E., Maurer, F.: Using Horizontal Displays for Distributed & Collocated Agile Planning. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 38–45. Springer, Heidelberg (2007)
3. IBM Jazz Server, `http://www-01.ibm.com/software/rational/jazz/` (last accessed March 10, 2009)
4. Rally Software, Rally's Agile Life Cycle Management Solutions, `http://www.rallydev.com/` (last accessed March 10, 2009)

The Future of Lean in an Agile World

Steven Fraser[1], Pekka Abrahamsson[2], Rachel Davies[3], Joshua Kerievsky[4], Mary Poppendieck[5], and Giancarlo Succi[6]

[1] Director, Cisco Research Center, USA
sdfraser@acm.org
[2] Professor, Helsinki University, Finland
pekka.abrahamsson@cs.helsinki.fi
[3] Coach, UK
rachel.davies@tiscali.co.uk
[4] Principal, Industrial Logic, USA
Joshua@industriallogic.com
[5] President, Poppendieck LLC, USA
mary@poppendieck.com
[6] Director and Professor, Bolzano Free University, Italy
Giancarlo.Succi@unibz.it

Abstract. Lean was first popularly exposed to the Agile Software development community with the publication of Mary and Tom Poppendieck's book "Lean Software Development: An Agile Toolkit" in 2003. While there has been much interest in Lean software practices, they have not attracted quite the same degree of popularity in the literature as other "Agile" practices. Why is this so? Is Lean harder to adapt and adopt – does Lean require a greater critical mass (team and system scope and scale) to make value tangible? What are the challenges that Lean is best suited to address? What are the areas for ongoing research and how should we proceed? – These are some of the questions that will be discussed and debated by this panel of leading practitioners and academics.

1 Steven Fraser *(Panel Impresario)*

STEVEN FRASER is the Director of the Cisco Research Center in San Jose California with responsibilities for developing and managing university research collaborations. Previously, Steven was a member of Qualcomm's Learning Center in San Diego, California with responsibilities for technical learning and development. Steven held a variety of technology management roles at BNR/Nortel including Process Architect, Senior Manager (Global External Research), and Design Process Advisor. In 1994, he was a Visiting Scientist at the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU) collaborating on the development of team-based domain analysis (software reuse) techniques. Fraser was the XP2006 General Chair, the Corporate Support Chair for OOPSLA'07 and OOPSLA'08, and Tutorial Chair for both XP2008 and ICSE 2009. Fraser holds a doctorate in EE from McGill University in Montréal - and is a member of the ACM and a senior member of the IEEE.

2 Pekka Abrahamsson

PEKKA ABRAHAMSSON is a full professor of computer science in University of Helsinki. He holds also an adjunct chief scientist s position in SINTEF, Norway. current responsibilities include managing a FLEXI-ITEA2 project, which involves 35 organizations from 7 European countries. The project aims at developing agile innovations in the domain of global, large and complex embedded systems development. His previous project was awarded an ITEA Achievement for outstanding industrial impact. His research interests are centred on business agility and lean software development, agile software development and empirical software engineering. He has coached several agile software development projects in industry and authored more than eighty scientific publications focusing on software process and quality improvement, agile software development and mobile software. He was a Nokia Foundation Award in 2007 for his achievements in software research.

Many position the lean thinking and agile development in the same school of thought. Still, the lean angle of agile development has received considerably less attention in research and practice. One reason for this may be the fact that many agile transformations are bottom-up endeavors. After successfully practicing agile for a period of time, companies tend to find an increasing pressure to move towards lean. It appears to be evident that the lean concepts communicate better to the managers and senior managers than those of agile. Yet, while the lean philosophy has existed for several decades, I would suggest that there is still room for novel interpretations of the lean thinking into the volatile software business environments. An interesting debate is ongoing whether the lean culture can be fully grasped by the western software world even though apparently this has been achieved in the manufacturing business. Answers are soon to come since e.g. European large software industry and research has begun its efforts to research and develop the lean capabilities and understanding further.

3 Rachel Davies

RACHEL DAVIES is an Agile Coach whose expertise is recognized internationally across the XP, Scrum, and DSDM communities. She has more than 20 years experience in software development and started her own agile journey in 2000 as a programmer in an XP team. Rachel has served on the board of directors of the Agile Alliance for 6 years. Rachel has presented at numerous conferences on topics related to agile coaching and has participated in the XP 200x conference program every year since 2001. She has written a book on Agile Coaching which is to be published by Pragmatic Bookshelf in the summer of 2009.

I've worked with a number of companies keen to implement a Lean approach. What I found was that the teams loved the Lean ideas they read about, but they got stuck in turning principles into practice. They needed practical advice on how to connect Lean thinking with Agile practices. It's also hard for teams to know what Agile practices don't map to Lean.

I've been surprised by the reaction in the Agile community to Kanban, which applies Lean principles of flow and limits work-in-progress. It seems that abandoning iteration planning activities is too extreme for some. My position on this panel is that Lean is not a panacea but can lead us to rethink how we apply agile in different contexts.

4 Joshua Kerievsky

JOSHUA KERIEVSKY leads Industrial Logic, a consultancy that guides organizations in Agile transitions. He is an expert in Extreme Programming and Agile Management, and a prolific author of eLearning literature on Refactoring, TDD and Patterns.

Lean is important to Agile Development because it focuses on Values rather than Practices. Lean suggests the importance of eliminating waste, yet it does not tell you how to do it. Thus, practitioners of Lean are free to invent ways to be lean, guided by lean values, which themselves are derived from solid and successful manufacturing realms.

I have found that the Lean terminology appeals to a vast number of people, particularly non-technical people. So it also offers an excellent way to connect Agile/XP concepts with people who are new to the movement.

The more I learn about Lean, the more I like it and see it as a great contribution to the Agile movement.

5 Mary Poppendieck

MARY POPPENDIECK has been in the Information Technology industry for over thirty years. She has managed software development, supply chain management, manufacturing operations, and new product development. She spearheaded the implementation of a *Just-in-Time* system in a 3M video tape manufacturing plant and led new product development teams, commercializing products ranging from digital controllers to 3M Light Fiber™. Mary is a popular writer and speaker, and co-author of the book *Lean Software Development*, which was awarded the Software Development Productivity Award in 2004. A sequel, *Implementing Lean Software Development*, was published in 2006. A third book, *Leading Lean Software Development*, will be published in late 2009.

In the 1950's, Toyota was scrambling to stay alive in a shattered economy. After a massive cash flow crisis triggered devastating layoffs, the company's leaders scrambled to conserve cash. There would be no more building of cars – or even car parts – unless there was a ready buyer. There would be no more wasting good parts on a bad car – bad quality would have to be discovered and its cause corrected immediately. There would be no more people specializing in a single job because this caused too much waiting – work waiting for specialists or specialists waiting for work. There would be no more wishful thinking when designing a new car – and no more re-inventing the wheel either. A vast amount of ingenuity was aimed at making every yen and every minute and every bit of experience count.

Now, more than ever, software development needs the same attitude. Lean is about doing things *just-in-time* rather than *just-in-case* – the only rational approach in hard economic times. It is about never letting defects into code, as opposed to finding them and fixing them later – and about rethinking what that really means, given today's technology. It is about leveraging the creative potential of every single person, and about never getting complacent – because the current approach just simply isn't good enough for survival. By its very nature, lean requires a systems perspective, because there is no such thing as partial survival.

Since lean boils down to focusing all assets on survival, lean ideas thrive in a rough economy or a tough competitive landscape – and it tends to falter when there is no tension, when there are too many politics, and when there is too much success.

6 Giancarlo Succi

GIANCARLO SUCCI is a Professor at the Free University of Bolzano-Bozen, Italy where he directs the Center for Applied Software Engineering. Before joining the Free University of Bolzano-Bozen, he was a Professor at the University of Alberta, Canada. He was also CEO of a small software company, EuTec. Succi holds a Doctorate in Computer EE from the University of Genoa and has been a registered Professional Engineer in Italy since 1991.

The Lean revolution has focused on two aspects: elimination of muda (waste) and the introduction of Jidoka (automation). Significant work has concentrated on the former, much less on the latter.

In particular, a plethora of software engineering tools are based on the ”push” paradigm: the user has to regularly run a tool to discover which rules are not fulfilled and to decide if an action is required. This occurs also in Agile environments. I claim that in software production we need to emphasize more the "pull" aspect of Lean, where the tools or the involved people performs the check automatically for the software engineer.

To my knowledge all proposals to insert Jidoka in software production relate to automated testing and continuous integration. I agree that automated testing and continuous integration (and the use of tools supporting it such as jUnit and Cruise-Control) implement the idea of Jidoka, but this is only one aspect of quality in software development. Also other quality attributes about the code produced and the process can and should make use of Jidoka during software production. Tools for non-invasive measurement can be instrumental to achieve this goal.

What Skills Do We Really Need in Agile Software Development? – Discussion of Industrial Impacts and Challenges

Minna Pikkarainen[1], Kieran Conboy[2], Daniel Karlstöm[3], Jari Still[4], and Joshua Kerievsky[5]

[1] VTT Technical Research Centre of Finland
minna.pikkarainen@vtt.fi
[2] NUI Galway
kieran.conboy@nuigalway.ie
[4] Golden Gekko Ltd
dk@goldengekko.com
[4] Jari Still
Jari.Still@f-secure.com
[6] Joshua Kerievsky
joshua@industriallogic.com

1 Summary of the Panel Discussion

Agile methods are largely used in software intensive companies in all over the world [1-3]. It seems that the use of agile methods have a high impact on the skills that are needed in software development [4]. For instance, in agile context, developers need to have capabilities to communicate all the information and continuously work as a part of the social teams (communication and agile [5]).

This panel discussion will be based on the research that focuses on understand impacts of agile software development on skills in software intensive organizations. The panel presentation will be done using the experienced of several companies doing agile software development. Based on the current experiences, it is revealed that:

1 The companies deploying agile methods do not usually have any training programs or plans for agile transition. This is the fact also in the situations in which the agile has been planning to be taken into use in large and globally distributed system development.
2 Due to the skill requirements, transition to agile seems to be quite painful process for many companies. The use of agile methods may cause an effect that developer's feel that their career progression does not exist anymore and that their importance as technical specialists is totally disappearing.
3 The use of agile methods causes also political issues that need to be taken into consideration when implementing the transition and agile based software development. Sociality increases contacts between the people who do not have communicated before and new issues may appear that have not been taken in the consideration before.

4 The bonus system of the companies needs to be changed. The developers should get the price of the work more as a team than as individuals which is the basic situation in many companies. Performance of the teams and companies should also be measured somehow and companies are seeking for good metrics how to do that.

2 Audience the Panel Discussion

It seems that the agile methods are now used also in the context in which they were not originally invented. They are quite common now also in the context of safety critical [6] [7, 8] large [9] and globally distributed [10] software development. This is creating challenges that directly affect on the work of the people and needed skills among the development teams and organizations.

Agile methods are also an important tool also for several consultancy companies who work in the field of software development. From their point of view, it would be significant to hear what kind of skills agile software development actually demands both in the teams and in organizations. As it seems that there is a lack of research related to the skills in agile software development context the topic would be also relevant for research organizations.

Attendees: industries: large and small; consultancy companies; researchers.

3 How the Panel Discussion Will Be Structured and Run?

The panel discussion will be held following. First each of the panellists will give a short presentation to the audience.

After the presentation the discussion will be held. The goal is first to present the challenges that attendants have found of this topic and then discuss of the results together with the whole group. The purpose of the approach is to reveal both challenges and solutions that attendants have related to the skills and agile software development.

During the discussions presenters will give examples based on the experiences of the companies that they have interviewed of this topic.

4 Bio of the Panelists

The panel discussion will be held together with the researchers and industrial people above.

Moderator
Minna Pikkarainen has graduated from the Department of Information Processing Science, University of Oulu and finished her PhD about the topic of improving software development mediated with CMMI and agile practices at 2008. Minna has been working as researcher and project manager in VTT Technical Research Centre of Finland more than 11 years now. During that time she has worked in 18 industrial

driven research projects doing close industrial collaboration with 8 organizations in Finland and in Ireland. Minna has participated as a key person for several large international ITEA project preparation work doing full project proposals and project outlines as collaboration together with large European level company networks (e.g. Flexi and Evolve projects). During 2007 – 2009 Minna has been leading VTT research group of the Large European projects called Agile ITEA (embedded agile software development) and Finnish consortium of ITEI (project about open innovations). Minna's research has been published in several journal and conference papers in the forums like ICSE, ICIS and Empirical Software Engineering Journal. So far Minna has provided agile trainings, workshops and invited talks for 10+ different industries related to agile methods. Minna has been member of Lero, The Irish Software Engineering Research Centre since 2006. For the past 4 years, her work and publications have been focused on research in the area of agile software development.

Kieran Conboy is a lecturer in information systems at the National University of Ireland Galway. His research focuses on agile systems development approaches as well as agility across other disciplines. Kieran is currently involved in numerous national and international projects in this area, and has worked with many companies on their agile initiatives including Intel, Microsoft, Accenture, HP, and Fidelity Investments. Kieran has chaired related conferences including the European Conference in Information Systems (Galway 2008) the XP and Agile Development Conference (Limerick 2008) and also has chairing roles at XP2009 and XP2010. Some of his research has been published in various leading journals and conferences such as the European Journal of Information Systems, the International Conference in Information Systems (ICIS), the European Conference in Information Systems (ECIS), IFIP 8.6 and the XP200n conference series. He is also associate editor of the European Journal of Information Systems. Prior to joining NUI Galway, Kieran was a management consultant with Accenture, where he worked on a variety of projects across Europe and the US. Kieran can be reached at kieran.conboynuigalway.ie.

Daniel Karlström is Chief Operating Officer and co-founder of Golden Gekko Ltd. a leading mobile application production company with operations in Europe, US and Asia. Daniel received a Ph.D. degree in Software Engineering from Lund University in 2004. His research interests include integrating agile practice with software product development. He has more than 15 years of experience in the software business both in software development and software process consulting all over the world. He has been involved in the agile community and conferences since 2000.

Jari Still is F-Secure Oy's Oulu Office site manager and the head of Mobile R&D. Still has been working at F-Secure since 2000. From 1991 to 2000 Still was working as CEO of Modera Point Oy. Before 1991 Still had worked with e.g. Nokia Mobile Phones and Siemens. Still is one of the founders of the Oulu Software Forum, and also has been acting as a Chairman of the Forum. Other posts have been e.g. Chairman of revontuliryhmä, Chairman of business development group in Oulu Innovation project and member of the Board in several companies. Related to Agile processes and product development Still is currently acting as a leader in FLEXI (ITEA) - project at F-Secure and he is also a Chairman of the management group of FLEXI

Finland -project. Before the FLEXI project on 2004-2007 Still started and lead Agile development at F-Secure in AGILE (ITEA) project.

Joshua Kerievsky leads Industrial Logic, a consultancy that guides organizations in Agile transitions. He is an expert in Extreme Programming and Agile Management, and a prolific author of eLearning literature on Refactoring, TDD and Patterns.

5 Past History of the Panel Discussion

The panel discussion will be based on both the research and industrial experiences related to the impacts of agile skills in software development.

References

[1] Schwaber, C., Fichera, R.: Corporate IT Leads the Second Wave of Agile Adoption (2005)

[2] Tan, C.H., Teo, H.H.: Training Future Software Developers to Acquire Agile Development skills. Communications of the ACM 50, 97–98 (2007)

[3] Ambler, S.W.: Survey Says..Agile Has Crossed the Chasm. Dr. Dobb's Journal: The World of Software Development 32, 59–61 (2007)

[4] Boehm, B., Turner, R.: Balancing Agility and Discipline. In: Balancing Agility and Discipline -A Guide for the Perplexed, p. 304. Addison Wesley, Reading (2003)

[5] Pikkarainen, M., Haikara, J., Salo, O., Abrahamson, P., Still, J.: The Impacts of agile practices on communication in software development. Empirical Software Engineering 13, 303–337 (2008)

[6] Fitzgerald, B., Hartnett, G., Conboy, K.: Customising Agile Methods to Software Practices at Intel Shannon. European Journal of Information Systems 15, 200–213 (2006)

[7] McCaffery, F., Pikkarainen, M., Richarsson, I.: AHAA -Agile, Hybrid Assessment Method for Automotive, Safety Critical SMEs. In: ICSE 2008, Leipzig, Germany (2008)

[8] Drobka, J., Noftz, D., Raghu, R.: Piloting XP on Four Mission Critical Projects. IEEE Software 21, 70–75 (2004)

[9] Lindvall, M., Muthig, D., Dasnino, C.W., Stupperich, M., Kiefer, D., Kähkönen, T.: Agile Software Development in Large Organizations. Computing Practices 37, 38–46 (2004)

[10] Layman, L., Williams, L., Damian, A., Bures, H.: Essential Communication Practices for Extreme Programming in a Global Software Development Team. Information and Software Technology 48, 781–794 (2006)

Perspectives on Agile Coaching

Steven Fraser[1], Erik Lundh[2], Rachel Davies[3],
Jutta Eckstein[4], Diana Larsen[5], and Kati Vilkki[6]

[1] Director – Cisco Research Center, USA
sdfraser@acm.org
[2] Principal, Compelcon, Sweden
erik.lundh@compelcon.se
[3] Coach, UK
rachel.davies@tiscali.co.uk
[4] Partner, IT Communications, Germany
jutta@jeckstein.com
[5] Partner, FutureWorks, USA
dlarsen@futureworksconsulting.com
[6] Manager, Nokia Siemens Networks, Finland
kati.vilkki@nsn.com

Abstract. There are many perspectives to agile coaching including: growing coaching expertise, selecting the appropriate coach for your context; and evaluating value. A coach is often an itinerant who may observe, mentor, negotiate, influence, lead, and/or architect everything from team organization to system architecture. With roots in diverse fields ranging from technology to sociology coaches have differing motivations and experience bases. This panel will bring together coaches to debate and discuss various perspectives on agile coaching. Some of the questions to be addressed will include: What are the skills required for effective coaching? What should be the expectations for teams or individuals being coached? Should coaches be: a corporate resource (internal team of consultants working with multiple internal teams); an integral part of a specific team; or external contractors? How should coaches exercise influence and authority? How should management assess the value of a coaching engagement? Do you have what it takes to be a coach? – This panel will bring together seasoned agile coaches to offer their experience and advice on how to be the best you can be!

1 Steven Fraser *(Panel Impresario)*

STEVEN FRASER is the Director of the Cisco Research Center in San Jose California (www.cisco.com/research) with responsibilities for developing and managing university research collaborations. Previously, Steven was a member of Qualcomm's Learning Center in San Diego, California with responsibilities for technical learning and development. Steven held a variety of technology management roles at BNR, NT, and Nortel including Process Architect, Senior Manager (Global External Research), and Design Process Advisor. In 1994, he was a Visiting Scientist at the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU) collaborating on the development of team-based domain analysis (software reuse) techniques. Fraser was the XP2006 General Chair, the Corporate Support Chair for OOPSLA'07 and OOPSLA'08, and Tutorial Chair for both XP2008 and ICSE 2009. Fraser holds a doctorate in EE from McGill University in Montréal - and is a member of the ACM and a senior member of the IEEE.

Steve's interest in Agile Coaching began with innovation coaching interventions in the mid-1990s and contributions to team methods of the FODA (Feature-Oriented Domain Analysis) software reuse process at the Software Engineering Institute (SEI) on the campus of Carnegie Mellon University (CMU) in Pittsburgh. As Director of the Cisco Research Center he leads the coordination of external research interactions between Cisco technical staff and university researchers – with a personal interest in software engineering related research.

2 Erik Lundh

ERIK LUNDH has more than 25 years experience in software development roles ranging from developer to R&D manager and board member. He is the Principal of Compelcon AB (www.compelcon.se), located in Helsingborg, Sweden. Company experience has included small, but mature innovation firms, start-ups large large organizations such as Ericsson and ABB. Erik had technical, 'Peopleware' [1] and organizational interests since the early 80s. Erik programmed military and industrial *just-in-time* (Lean) applications in the 1980's, was a "process and management guy" in the 1990's, and spent the fun part of the 2000's as an agile evangelist and coach. Erik is a core member and organizer of SPIN communities in Sweden and contributes to agile standards work through the IEEE. Erik has participated in all of the XP200x series of conferences (since 2000). In 2006 Erik was invited to Ericsson's first major agile transformation of 2300 R&D people at 10 sites in 5 countries. Erik currently works with executives in the Ericsson group on agile transformation initiatives where the internal agile coach is a key player. Erik is a founder of the international Agile Coaches Guild (www.agilecoachesguild.org).

In 2000, I suddenly became an agile coach. I had *talked about* agile since 1999 with clients and at conferences with the intention to host a few experienced (XP) coaches. We intended to build R&D Centers using agile, but dire times struck in 2000. My clients told me to walk the agile talk myself – instead of hiring help. I did – coached my first team – had instant success, and found that the outer form of XP's 'novice rules' [2] married well with my insights from just-in-time, patterns and 'Peopleware'[1]. My position on agile coaching is to create success and move on. My family life did not allow for onsite full time long-term gigs, and I discovered that it made me much more effective as a coach, but also distinctly transitional.

My teams attained self-sufficiency quickly so I was able to engage many teams and companies. I am proud that most of 'my' teams turned into successes, first in months, later in weeks. I learned to recognize agile failure scenarios, which I could turn around and which I should avoid. When I was invited to participate in the first agile transition at Ericsson, it became evident that we had to grow a competency of agile coaching. We started in recruitment in 2007 and founded our first international group of 30 agile coaches. At XP2008 I was thrilled to learn that companies like Nokia has started on similar journeys – growing their own cadre of agile coaches.

Many R&D and IT organizations are challenged by their business and work environments. From my perspective, agile coaches should offer people guidance on balance and a better work-life. Agile coaching is sometimes confused with facilitation or even technical, project and management work. In my opinion – to be an Agile Coach is a calling – not a meal ticket. However, in dire times, agile coaches may need to rely on broader set of core competencies, since soft roles are prime targets for corporate cost reductions.

1. DeMarco, T., Lister T.: Peopleware: Productive Projects and Teams. New York: Dorset House, 1st edition 1987. (ISBN 0-932633-43-9). 2nd edition in 1999, (ISBN 0-932633-43-9)
2. 'The Dreyfus Model of Skills Acquisition' In: Hunt, A.: Pragmatic Thinking and Learning, chapter 2 "Journey from Novice to Expert", pp. 29-56. Pragmatic Bookshelf, 2008, (ISBN 1-934356-05-0)

3 Rachel Davies

RACHEL DAVIES is an Agile Coach whose expertise is recognized internationally across the XP, Scrum, and DSDM communities. She has more than 20 years experience in software development and started her own agile journey in 2000 as a programmer in an XP team. Rachel has served on the board of directors of the Agile Alliance for 6 years. Rachel has presented at numerous conferences on topics related to agile coaching and has participated in the XP 200x conference program every year since 2001. She has written a book on Agile Coaching which is to be published by Pragmatic Bookshelf in the summer of 2009.

In writing my book about Agile Coaching, I have spent a lot of time examining what I do when I coach Agile teams. The key for me is not to solve problems for teams; instead the teams need to do the work.

As a coach, I show them alternatives and help them to reflect about what to do next. This is very different to what I see Scrum Masters doing. Too many organizations seem to think being Agile is about planning and tracking every move of the team. Doing this prevents the teams from taking responsibility for their own actions. The role of an Agile Coach is to help the team think for themselves and not to remove impediments and micro-manage the team's every move. I believe that organizations need to accelerate agile transitions with Agile Coaches not Scrum Masters.

4 Jutta Eckstein

JUTTA ECKSTEIN Jutta Eckstein, a partner of IT communication, is an independent consultant, coach and trainer from Braunschweig, Germany. Her know-how in agile processes is based on over ten years experience in developing object-oriented applications. She has helped many teams and organizations to make the transition to an agile approach. She has a unique experience in applying agile processes within medium-sized to large distributed mission-critical projects. This is also the topic of her already published book ‘Agile Software Development in the Large’ and of the one she’s currently writing on ‘Distributed Agile Software Development’. Besides engineering software she has been designing and teaching OT courses in industry. Having completed a course of teacher training and led many 'train the trainer' programs in industry, she focuses also on techniques which help teach OT and is a main lead in the pedagogical patterns project. She has presented work in her main areas at ACCU (UK), JAOO (Denmark), OOPSLA (USA), XP (Europe) and Agile (USA).

I became a coach after several years of being a developer both on- and off-site. At the time I started working for ParcPlace I was at first used to be called-in as a "fire-fighter" for our customers. After a while some of these customers decided to call me in before their fires started.

Every team and every team member carries a wide range of good and bad practices concerning software development processes. As a coach it is very important to mine this knowledge and use it for defining the team's own process. Thus it is much more important to respect the experiences of the team than the experiences of some process methodologist. Of course the knowledge of colleagues and other process methodologists is a great source for filling the gaps in the team's own process and for improving it. Regular project retrospectives after really short iterations help to find these gaps and the necessary improvements. As soon as the team trusts the process and knows how to make any necessary changes the coach needs to step out – since the team can only organize itself when it really gets the responsibility for doing so. The coach must lead the team towards self-organization – this is achieved by leaving the team as soon as possible.

If you are looking for a coach, I would absolutely recommend one with strong social skills rather than simply technical skills. I have never seen a project fail because of technical reasons. Thus the coach has to be a catalyst, a facilitator, an ombudsman, a team player, and someone able to make hard decisions.

5 Diana Larsen

DIANA LARSEN is a senior consultant and partner at FutureWorks Consulting (www.futureworksconsulting.com) in Portland, Oregon. Diana consults with leaders and teams to create work processes where innovation, inspiration, and imagination flourish. With more than fifteen years of experience working with technical professionals, Diana brings focus to the human systems of organizations, teams and projects. She activates and strengthens her clients’ proficiency in shaping an environment for productive teams and thriving in times of change. Diana co-authored "Agile Retrospectives: Making Good Teams Great!" and writes articles and occasional blog posts. Currently serving as chair of the Agile Alliance Board of Directors, she co-founded the Agile Open Northwest conference and the international Retrospective Facilitators Gathering.

My role with XP/Agile projects falls into the categories of consulting to the organization and coaching the coaches. My first team coaching assignment happened in 1965 when, as a teenager, I progressed from managing editor of my high school weekly newspaper to editor-in-chief, then past editor. As past editor, I still had an active role to play though I was no longer writing or editing news articles, features stories, or editorials. I became a mentor and coach to the whole newspaper staff. As the first of scores of team coaching assignments I would step into over the next 40+ years, that experience signaled the beginning of my fascination with people working in groups – whether teams or organizations – and how those groups develop into strong, effective actors in their environments or spiral into dysfunction. Now I mentor others who have that same passion to see their teams acquiring the mysterious, powerful synergy that we only find when we work well together.

Agile coaches focus on the total health of the team, including healthy performance, productivity, and working relationships. Many people come well equipped to foster health in one or two of the three areas. Few show strength at all of them. Agile coaching requires daily, pragmatic attention to individual contributions, as well as an understanding of the team as a whole system functioning within a larger system. Even more than others, Agile coaches deal with the issues of ambiguity, transparency, and paradox; bringing new perspectives and frames, as well as guidance on approach, process and practices. Their value to the team, and to the enterprise, lies in their ability to support the team in sticking to agreements, making the most of feedback of all kinds, finding areas of continuous improvement, adding value to the product, and staying accountable. They direct, guide and evaluate the team's progress when needed, yet know when to step back, follow, and encourage shared leadership among team members too.

It's a tough role that spans a wide range of skills, behaviors, natural aptitudes, and business knowledge, in addition to an ability to function in the "threshold" space—at once a valued part of the team, standing outside the team, and working as a liaison between the team, its customers and stakeholders, and the organization.

6 Kati Vilkki

KATI VILKKI heads the Agile Coaching team at Nokia Siemens Networks (NSN). Since Kati joined Nokia in 1994 after completing her Masters in Computer Science from Helsinki University, she has held various management and development positions mostly within product development. Kati also has a broad experience in organizational and leadership development. She has headed quality and process management, managed software process and other improvement programs, driven operational mode development and has been the change agent for many different projects. Starting in 2005 one of Kati's key change programs was the agile transformation of an NSN environment with very large-scale programs and an organization which had a long tradition of "waterfall" development modes. As a result of the successful changes – agile development is now firmly established in the software process.

In my definition an agile coach is someone who "lives with the organization - team"; they coach for a extended period of time in comparison to a consultant who often has a looser and shorter relationship with the team. In agile transformation we need trainers, consultants, facilitators, change agents – and above all – agile coaches. Ideally a coach should be able to work in all these capacities: train, demonstrate new practices and facilitate organizational learning. In my opinion, facilitating learning is the coach's most important contribution.

Agile transformation requires learning in many areas: agile thinking and change in the mind-set, how to use the different agile practices and methods, how to achieve a paradigm shift, and how to change the organizational and management culture. Very few coaches have all needed skills – from coaching a team of developers to use XP practices – to what kind of changes are required in an organization of several thousand people. Even if a coach is familiar with all areas of required change – the organization most often requires several coaches; some people are more credible with leadership teams and others with development teams. Once an organization puts "coach" in a "team coach" or a "management coach" box, it is very difficult for a coach to escape. Agile coaches need to have different backgrounds and they need to work in close cooperation with others.

My best experience coaching was with a combination of internal and external coaches. Externals provide an outside view and have a different kind of credibility, which comes from working with a diversity of organizations. Internal coaches know the local circumstances and the people – working in pairs or coaching teams both views can be utilized. Ideally one should never coach alone – with a pair or a team of coaches – the learning opportunities are much richer.

Author Index