Mälardalen University Press Licentiate Thesis
No.136

Software Testing in Agile Development

Technological and Organisational Challenges

Adnan Čaušević

June 2011

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN 1651-9256
ISBN 978-91-7485-015-4
Printed by Mälardalen University, Västerås, Sweden

Abstract

The contemporary industrial trend towards agile software development brings forth new concerns, challenges as well as opportunities. One of the main issues concerns the quality of the final product, for which testing is the well-known assurance mechanism. However, how to perform testing using existing expertise in an agile environment presents a challenging issue for the software industry. This can potentially create confusion and contra productivity which can lead to a situation where testing teams and their practices are considered obstacles for the full implementation of agile processes within an organisation.

This thesis identifies and addresses test-related organisational and technological challenges in an agile environment. In this context, we propose a new role for traditional testers which enables them to integrate with the agile team as well as fully exploit their knowledge in the new context. We have conducted an elaborate industrial survey on the preferences and practices with respect to the contemporary aspects of software testing, and identified test-driven development as an important technological area for improvement. A subsequently performed systematic review on empirical evidence related to test-driven development revealed a list of factors which may limit its widespread industrial acceptance and usage. Knowledge of testing was identified as one of those factors and we further attempted to confirm its significance through a controlled experiment performed with master students.

Our future works aim to confirm these research findings in wider as well as industrial settings, and investigate other limiting factors in detail, with the aim of providing guidelines for achieving better utilisation of testers and testing practices.

Acknowledgements

This thesis could not have been done without the great support of my supervisor Sasikumar Punnekkat and my co-supervisors Daniel Sundmark and Ivica Crnković. Thank you guys for your leadership, patience and knowledge you shared so unselfishly. Even though this thesis was my destination, your supervision made me realise how the journey itself mattered the most.

As a Ph.D. student I was relying on my supervisors support in publishing research results, but co-authoring with researchers out of my comfort zone greatly improved my collaboration and interaction skills. Indeed this is something I am very thankful for to Abdulkadir Sajeev, Rikard Land, Frank Lüders, and Iva Krasteva.

Travelling to conferences and research project meetings is another link in the chain of experience a graduate student should have. Thank you Stig Larsson, Sigrid Eldh, and Radu Dobrin for being an often travel companion in this phase of my study. Mingling is so much easier with you guys around.

When not travelling, I had to share my office space with really great roommates: Srinivasan Jayakanth (JK), Stefan Björnander, Kathrin Dannmann, Etienne Borde, Aleksandar Dimov, Andreas Johnsen, Vijayalakshmi Saravanan (Viji), Hüseyin Aysan, Abhilash Thekkilakattil, and Jiale Zhou. Thank you guys for being silent, but also cheerful and always ready for a small talk.

It's not very easy to focus on the research when there are administrative issues hanging above your head. Luckily, I had administrative people around me to always rely on. Thank you Harriet Ekwall, Gunnar Widforss, Monica Wasell, Susanne Fronnå, Carola Ryttersson, and Malin Rosqvist.

Directly or indirectly, many senior researchers at MDH have provided help to my Ph.D. studies. Thank you Hans Hansson, Kristina Lundqvist, Paul Pettersson, Cristina Seceleanu, Thomas Nolte, Dag Nyström, Damir Isović, Jan Carlson, and Tiberiu Seceleanu mostly for isolating me from the world of funding but also for having the time for me and my questions.

As a person I am very dependant on the communication and interaction with other human beings, and MDH could not be a better choice to get many interesting discussions during the coffee breaks, travels or other social events. Thank you Ana Petričić, Ana Živković, Aneta Vulgarakis, Antonio Cicchetti, Barbara Gallina, Batu Akan, Branka Pavetić, Farhang Nemati, Federico Ciccozzi, Giacomo Spampinato, Hongyu Pei-Breivold, Jagadish Suryadevara, Josip Maraš, Juraj Feljan, Leo Hatvani, Luka Ledinički, Mehrdad Saadatmand, Mikael Åsberg, Moris Behnam, Nikola Petrović, Rafia Inam, Saad Mubeen, Séverine Sentilles, Stefan Bygde, Svetlana Girs, Thomas Leveque, and Yue Lu for sharing a few moments of your life with me.

I would like to express my gratitude to my parents Zuhdija and Šefika Čaušević as well as to my sister Azra Čaušević for their unconditional support and love through all of my life. I appreciate your smile and understand your tears.

I would like to thank to my wife Aida Čaušević for supporting me and believing that I can achieve much more. If I have to start this journey again I could not imagine anyone else beside me except you. I love you!

And last, but most certainly not the least, I would like to thank to my daughter Alina Čaušević for making me a complete person. Your smile, your bite, your hug, your cry... everything of yours helps me move forward. I love you, too.

Adnan Čaušević
Västerås, June 21, 2011

List of Publications

Papers Included in the Licentiate Thesis[1]

Paper A *An Industrial Survey on Contemporary Aspects of Software Testing*, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, In proceedings of the International Conference on Software Testing (ICST), Paris, France, April 2010

Paper B *Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review*, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, In proceedings of the International Conference on Software Testing (ICST), Berlin, Germany, March 2011

Paper C *Impact of Test Design Technique Knowledge on Test Driven Development: A Controlled Experiment*, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, In submission

Paper D *Redefining the role of testers in organisational transition to agile methodologies*, Adnan Čaušević, A.S.M. Sajeev and Sasikumar Punnekkat, In proceedings of International Conference on Software, Services & Semantic Technologies (S3T), Sofia, Bulgaria, October, 2009

[1] The included articles are reformatted to comply with the licentiate thesis specifications

Other relevant publications

Conferences, Workshops and Poster Sessions

- *Reuse with Software Components - A Survey of Industrial State of Practice*, Rikard Land, Daniel Sundmark, Frank Lüders, Iva Krasteva and Adnan Čaušević, International Conference on Software Reuse, Springer, Falls Church, VA, USA, September, 2009

- *A Survey on Industrial Software Engineering*, Adnan Čaušević, Iva Krasteva , Rikard Land, A.S.M. Sajeev and Daniel Sundmark, Poster session at International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP2009), p 240241, Springer, Sardinia, Italy, Editor(s):P. Abrahamsson, M. Marchesi, and F. Maurer, May, 2009

Technical Reports

- *An Industrial Survey on Software Process Practices, Preferences and Methods*, Adnan Čaušević, Iva Krasteva , Rikard Land, A.S.M. Sajeev and Daniel Sundmark, MRTC report ISSN 1404-3041 ISRN MDH-MRTC-233/2009-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, March, 2009

Contents

I

Thesis

Chapter 1

Introduction

Traditional software development life cycle has become inadequate to preserve quality of software products when organisations attempt to shorten their time-to-market. In many cases the quality control is often reduced or postponed due to the reduced deadlines or overrun of the development phase [1] [2]. Organisations are in need of a new process that will value quality in each stage of their product development without interfering with the product delivery schedule. They are increasingly turning their interest to agile methodologies [3].

Agile is, indeed, a software development philosophy that will battle with short delivery schedules by creating a product with fewer features instead of lowering quality standards of the same product. The problem is that many proven cases of agile development in large scale environment are specific to each organisational setting and their best practices cannot be easily implemented within another organisation. Of course, at the same time, we can only guess the number of unsuccessful agile development attempts in organisations, without publicly available reports on their failures (in literature known as publication bias [4]). However, during our involvement in FLEXI, an EU-ITEA2 funded Project [5], we became aware from our industrial partners, of many of issues related to the transition from the traditional lifecycle to the Agile approach. One of the reason for such issues, could be in fact that organisations are trying to reuse techniques and tools from traditional development process that may not be applicable within particular agile practices, and blamely Agile development processes may not be fully justifiable.

The research presented in this thesis, originated from such a premise and investigates if traditional approaches to software testing with existing practices in place could be utilised to full extent within agile development.

1.1 Background

In this thesis we will be using several concepts from three different areas, viz., Agile development, Software testing and Test-Driven Development. We now present some key concepts from these areas, before providing the details on the contributions of this thesis.

1.1.1 Agile Development

Agile development is considered a relatively young software engineering discipline that emerged from industrial needs for a software development process where the main focus should be on the customer and their business needs. The idea is to have a constant communication channel with the customer by iteratively providing working software product with currently most needed business values built in. Historically, the idea behind an agile approach is actually not new. It was reported [6] that NASA Project Mercury (first US human spaceflight program in 1960s) used time-boxed iterations with tests written before each increment - an activity very similar to what is known today as a test-driven development (TDD).

Agile is not a software development process by definition, but rather a philosophy based on a set of principles. These principles are listed in the so called "Agile Manifesto" [7]. Since understanding of agile is relying on those twelve principles, we are listing them here:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers must work together daily throughout the project.*

5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7. *Working software is the primary measure of progress.*
8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9. *Continuous attention to technical excellence and good design enhances agility.*
10. *Simplicity - the art of maximizing the amount of work not done - is essential.*
11. *The best architectures, requirements, and designs emerge from self-organizing teams.*
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Agile Manifesto [7]

By following these principles, organisations are committing to have a continuous feedback with customer and provide value to their business needs.

Several software development processes use some of those principles, like: eXtreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Feature Driven Development (FDD), etc. usually referring to them as *agile software development methods*. Aside from following agile principles, each of those methods contains different *agile practices*. Pair programming (PP), test-driven development (TDD) and continuous integration (CI) are just a few to mention.

An overview of one Scrum iteration (sprint), as an example of agile development process, is shown in Figure 1.1. Prioritised product backlog is used to select user stories for the upcoming sprint. By dividing them into concrete tasks, they become part of the current sprint backlog. During the period of 2-4 weeks only items in the current sprint are completed on a daily basis. After each sprint a potentially shippable product increment should exist.

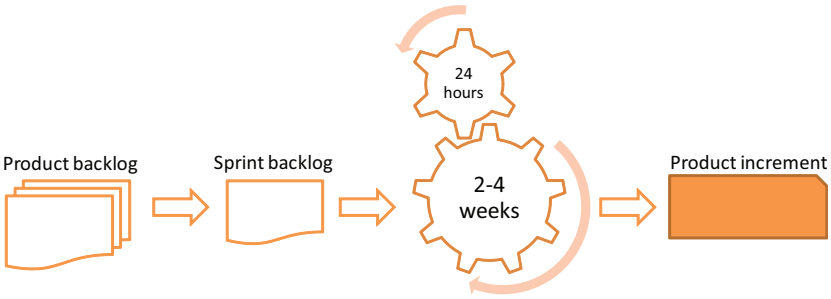

Figure 1.1: One sprint overview in Scrum process

1.1.2 Software Testing

Software testing is a major activity in software development and has two main goals:

- **to confirm** a software solution is behaving as per its requirements, and
- **to find faults** in a software which are leading to its misbehaviour.

It is important to note how testing cannot be used as a proof of fault free software. A famous quote from Edsger Dijkstra [8] is describing this as: “Testing can only show the presence of errors, not their absence”. One of the reasons why we cannot claim there are no faults in software is in fact that exhaustive testing of any, especially complex systems, is just not possible due to the high number of variables influencing its final outcome.

Commonly, there are three levels of testing of software systems [9]:

- **System level** - has the purpose of testing overall system functioning from a user perspective.
- **Integration level** - has the purpose of testing interconnections between various components/modules during their integration phase.
- **Unit level** - has the purpose of testing functional and non-functional properties of a single unit/module/component of the system.

Software testing is a widely researched domain of its own with a multitude of techniques and tools proposed for industrial practice. A comprehensive discussion on this vast research domain is beyond the scope of this thesis and hence not attempted.

1.1.3 Test-driven development

Test-driven development (TDD), sometimes referred as test-first programming, is a practice within the extreme programming development method proposed by Kent Beck [10]. TDD requires the developers to construct automated unit tests in the form of assertions to define code requirements before writing the code itself. In this process, developers evolve the systems through cycles of testing, development and refactoring. This process is shown in Figure 1.2.

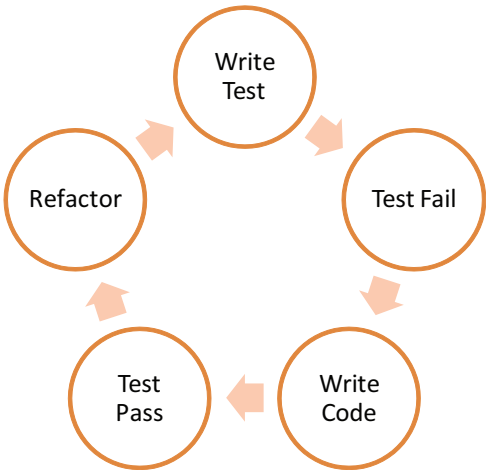

Figure 1.2: Test-driven development practice overview

In their experiment, Flohr and Schneider [11] prescribed TDD activities to students as a list of next activities:

1. *Write one single test-case*
2. *Run this test-case. If it fails continue with step 3. If the test-case succeeds, continue with step 1.*
3. *Implement the minimal code to make the test-case run*
4. *Run the test-case again. If it fails again, continue with step 3. If the test-case succeeds, continue with step 5.*
5. *Refactor the implementation to achieve the simplest design possible.*

> 6. *Run the test-case again, to verify that the refactored implementation still succeeds the test-case. If it fails, continue with step 5. If the test-case succeeds, continue with step 1, if there are still requirements left in the specification.*
>
> Flohr and Schneider [11]

1.2 Motivation and Problem Description

Today's business needs are demanding from software organisations to accept a constant pace of change as it reflects the current market and economic demands. According to the agile philosophy delivering an evolving software product without having a predefined set of requirements that will be changed at a later stage is something companies should not fight against, but rather embrace. Agile software development is one representative of the current industrial solutions to this challenge.

But this comes with a price. Adopting agile development for many organisations creates not only a phase shift in thinking on how to develop software but it also introduces significant amount of changes to their daily activities [12]. These changes consist of facilitating continuous product integration, ability to prioritise tasks, committing to its delivery all the way through daily stand-up meetings and burn-down charts.

In particular, changes affecting testing teams and testers may create additional confusion with respect to understanding who is responsible for the product quality and how to allocate time for this activity. In agile development, quality is everyone's responsibility and having in mind that traditional testing can consume even more than 50% of the total development time [9], testers do have a concern of ensuring how this time will be allocated in agile development.

1.3 Outline of thesis

This thesis consists of two main parts. The first part is organised as follows: Chapter 2 presents a summary of the research conducted with description of the research process and its major contributions. Chapter 3 provides related work with respect to both technological and organisational perspectives of our research. Thesis conclusion and guidelines for future work are outlined in Chapter 4. The second part of the thesis consists of Chapters 5 through 8 which represent research publications included in this thesis.

Chapter 2

Research Summary

Overall goal of our research efforts is:

to identify deficiencies in current testing practices in agile development environments and provide validated methods of better utilization of testers and testing techniques.

In order to help organisations successfully utilise agile practices, we set out to investigate how well software testing fits with the state of the practice of agile philosophy or the agile manifesto. The goal of this research could be viewed from two dimensions:

- **Technological**, defined with the top-level research question:
  RQ-1: *What are the technological challenges of traditional software testing in agile?*
- **Organisational**, defined with the top-level research question:
  RQ-2: *What are the organisational challenges of traditional software testing in agile?*

From the technological perspective, the goal is to identify test related practices, methods, techniques, improvements or practice adoptions which will provide most benefit to an organisation. It is also required to identify limiting factors for usage of such practices in an industrial environment.

From an organisational or process point of view, the goal is to define a new role for testers during an organisational transition towards agile methodology. It is our belief that this role will enhance the stature of testers as well as enable the company to effectively deploy the testers in the new environment.

2.1 Research Methodology

The research is based on empirical methodologies including analysis of qualitative and quantitative data. Literature and industrial surveys were performed in order to perceive the state of the art and state of the practice. Experiences from industry on this topic were collected and summarised with the research in a reusable form on a higher level of abstraction intended to be provided as guidelines for transition organisations.

2.2 Research Process

In Figure 2.1 an overview of the conducted research process is presented.

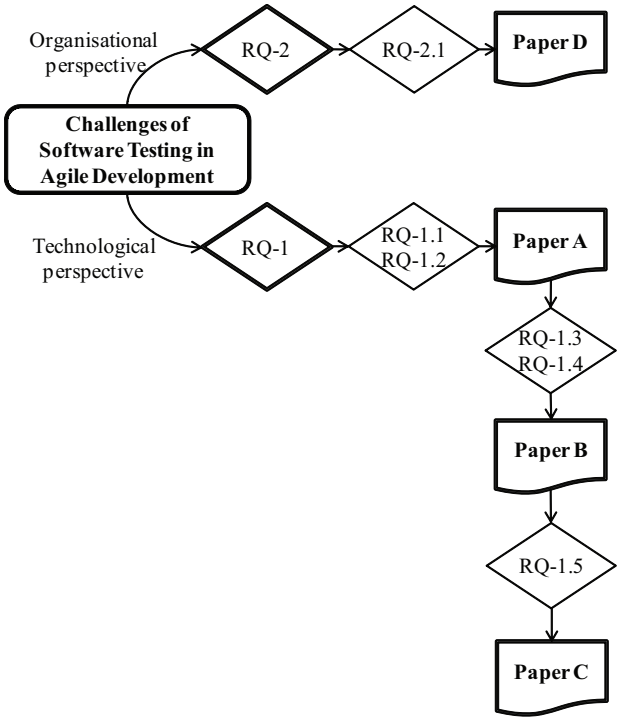

Figure 2.1: Research process overview

2.2.1 Technological perspective

As a starting point in detailed investigation of the top-level research question within the technological perspective (RQ-1), we decided to start the process by forming next research question:

RQ-1.1: *What are the current industrial preferences and practices related to the contemporary trends on software testing?*

To address this question, we decided to join our effort with several other researchers in order to define and execute a questionnaire through an online web based survey [13]. With this survey we specifically targeted industrial opinion on the usage of the current and preferred industrial practices and methods on software testing. During the formulation, execution and analysis of this empirical study, the subsequent research question evolved as:

RQ-1.2: *Can we identify the factor in which the preference and practice show maximum difference?*

After analysis phase was performed on the collected data, out of 22 examined test related practices, test-driven development (TDD) gained the highest score of "dissatisfaction". This means that among the respondents, the accumulated absolute difference between the preferred and the actual level of usage of TDD was highest. Further analysis revealed that the preferred level of usage of TDD was significantly higher than the actual level at which it has been practised. This result was interpreted as "Respondents would like to use TDD to a significantly higher extent than they actually do presently". This was an interesting finding for which we could not provide any clear and obvious reasons why this situation exist in industry. In order to get the broader view of the problems related to usage of TDD, the next research question was formulated as:

RQ-1.3: *How can we get a deeper insight on the factor with maximum difference?*

Realising that TDD as a practice should be investigated further we had to make a decision on how to proceed with the research process. One alternative was to further investigate industrial opinions by performing directed interviews with selected organisations. Another could be to organise a new questionnaire

survey with specific and directed questions relating to the usage of TDD. The problem with those solutions was that they are all providing an industrial perspective to the usage of TDD which we to some extent already gained from our first survey. We thought that academic opinions on the usage of TDD should also be considered in our research since after looking at some initial search results we noticed a growing number of empirical publications directly investigating benefits of TDD. For those reasons we decided to perform a systematic literature review on empirical studies of TDD.

When completed, the systematic literature review brought forward a list of 48 empirical studies on TDD, conducted in academic, industrial or mixed settings. Study participants were students as well as professionals. This result lead to forming a new research question:

RQ-1.4: *Can we identify and list limiting factors of TDD from the results of the literature study?*

Empirical studies, identified in the systematic literature review, were performed with different experiment designs (number of participants, complexity of problems, duration of study, etc.) making it difficult to directly compare the findings and easily create a common conclusion. We decided to identify and list all negative, neutral or positive effects of or on TDD and group them in common effect areas. Especially, we noted effects of TDD with explicit claims on requirements for a successful usage of TDD. In order for effect area to be considered as a limiting factor, next criteria had to be fulfilled:

- The effect area had to contain at least two studies with observations of negative effects of or on TDD
- The effect area had to contain more studies with observations of negative effects of or on TDD than studies with observations of positive effects of or on TDD
- Negative effects in the effect area had to be observed in at least one study performed in an industrial setting

Applying those criteria on selected research publications identified and listed seven potential limiting factors of industrial adoption of TDD: increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code.

Out of these seven factors, we decided to explore one factor in detail to confirm its impact and see what kind of guidelines could be provided. "Lack of testing knowledge" came as the first obvious choice due to our own research leanings as well as due to the potential for independent exploration and perceived impact. The next research question was formed as:

RQ-1.5: *Can we confirm significance of testing knowledge as a limiting factor for TDD adoption?*

During the autumn of 2010 a controlled experiment with master students was performed as part of the course on Software Verification and Validation provided by Mälardalen University. The objective of the experiment was to investigate if developers who were educated on general testing knowledge will be able to utilise TDD more effectively. As a result of the experiment we noticed that students had difficulties creating negative test cases.

2.2.2 Organisational perspective

In order to perform detailed investigation of the top-level research question within the organisational perspective (RQ-2), we setup the next specific research question:

RQ-2.1: *What to do with traditional testing department when an organisation transits to agile development process, where tester's roles seems to be ambiguous and diminished?*

In this investigation we considered several options for traditional software testers during their organisation's transition towards agile software development. Among various alternatives we proposed a new role of: "Project Mentor" for testers. With this role we wanted to emphasise testers ability to communicate with development team on technical aspects of software development while at the same time being able to recognise the value for the customer by understanding the overall functional behaviour of the system.

2.3 Contribution

Since the thesis is written as a collection of papers, its contributions are summarised with contributions from each individual research paper. Relation between research paper contribution and research questions is presented in Table 2.1.

	Paper A	Paper B	Paper C	Paper D
RQ-1	✓	✓	✓	
RQ-1.1	✓			
RQ-1.2	✓			
RQ-1.3		✓		
RQ-1.4		✓		
RQ-1.5			✓	
RQ-2				✓
RQ-2.1				✓

Table 2.1: Relation between research questions and publications

2.3.1 Paper A

An Industrial Survey on Contemporary Aspects of Software Testing, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, In proceedings of the International Conference on Software Testing (ICST), Paris, France, April 2010

Summary Using data from an industrial survey [13] a state of the practice paper was written. The survey in addition to confirming some popular beliefs also lists several noteworthy findings from the perspectives of respondent categories such as safety-criticality, agility, distribution of development, and application domain. These findings clearly depict negative discrepancies between the current practices and the perceptions of the respondents. This paper covers RQ-1.1 and provide contribution to RQ-1.2 by identifying test-driven development (TDD) as a factor with maximum difference between current and preferred practice.

My contribution I was the main author of this paper contributing with data analysis (performed using custom made software, developed by me for this

purpose). Co-authors supervised the process and helped in formulating findings and descriptive statistics.

2.3.2 Paper B

Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, In proceedings of the International Conference on Software Testing (ICST), Berlin, Germany, March 2011

Summary As a direct result of investigation from Paper A, a systematic literature review on TDD was performed. After initial keyword search on seven major research databases, results yielded 9462 publications. In several steps we removed publications that are not of an interest having 48 publications as the final number of our systematic review. With this activity RQ-1.3 was addressed. The process of extracting effects areas on or of TDD from selected research publications and identifying limiting factors contributed to RQ-1.4. Seven limiting factors were identified viz., increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code.

My contribution I was the main author of this paper contributing in obtaining collection of papers from the search databases, filtering and removal as well as analysis of findings presented in selected collection of papers. Co-authors helped to filter the papers and also performed reading of selected list of publications to validate the findings.

2.3.3 Paper C

Impact of Test Design Technique Knowledge on Test Driven Development: A Controlled Experiment, Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat, (In submission)

Summary Among the seven limiting factors identified from the systematic study in Paper B, knowledge of testing was selected to be further investigated as part of a controlled experiment with master students in order to address research question RQ-1.5. The experiment was designed around course on Software Verification and Validation at Mälardalen University. Participants were

divided into two groups solving two problems on two different occasions, before and after the course. The analysis was performed on the collected source code and test scripts created by students, as well as questionnaire survey responses. Results are showing positive improvements of test code coverage but no statistically significant difference exist between pre- and post- course groups. Qualitative analysis of data revealed lack of negative test cases resulting in students inability to detect bugs related to unspecified behaviours.

My contribution I was the main author of the paper, contributing in setting up the pre-requirements for the experiment (lab instructions, problems user stories, SVN, etc.), collecting data points and performing the analysis. Co-authors helped in study design, analysis of the data and in writing section on statistical analysis.

2.3.4 Paper D

Redefining the role of testers in organisational transition to agile methodologies, Adnan Čaušević, A.S.M. Sajeev and Sasikumar Punnekkat, In proceedings of International Conference on Software, Services & Semantic Technologies (S3T), Sofia, Bulgaria, October, 2009

Summary This paper provides a state of the art analysis of tester role in Agile organisation and propose a new role called “Project Mentor”. A major task of project mentors is to manage the expectations of the customers and other stake holders. This requires domain knowledge and the ability to speak in the language of the customers, which often programmers lack. Similarly, for managers, recognising the limitations of programmers is also a difficult task. Managers without a technical background often fail to understand difficulties which are faced by programmers on a daily basis. Testers as project mentors, we believe, will be in a position to better appreciate these difficulties and translate them to other stake holders with the help of their domain knowledge. A mentor’s role of helping others to implement quality in their daily activities could contribute significantly to the success of the project. This paper directly address research question RQ-2.1.

My contribution Idea for this paper originated from a discussion with visiting professor Abdulkadir Sajeev. I was the main author of this paper but the writing process was an iterative contribution of all authors.

Chapter 3

Related Work

Since our research is based on challenges from two fairly different perspectives, technological and organisational, we are presenting here related work from both of them independently.

3.1 Technological perspective

Agile does not have a formal definition behind its processes which makes it very hard for academic researchers to measure the quality impacts it can produce and in specific to reason about its claimed success. What researchers can do is to perform a series of empirical studies in academic or industrial settings for the purpose of evaluating quality improvements introduced with agile methodologies. Another aspect of investigation about agile development are the growing number of claimed success stories from industry that are presented to the community. By contributing with their experience and lessons learnt from projects with varying size and duration, industry is making a significant impact on the current body of knowledge that should not be neglected.

The central research paper on agile methodologies is ”Empirical Studies of Agile Software Development: A Systematic Review” [14]. This systematic literature review provides information regarding up to date findings w.r.t. empirical evidence of agile software development. It also provided additional insights for our own systematic literature review of empirical studies on TDD. Another additional resource on general understanding of agile methods is a chapter of Williams [15] within Advances in Computers book series where she describes different agile principles, practices and methodologies.

3.1.1 Empirical Studies on TDD

Several publications with empirical finding were also used in our research. In this section we are grouping them by the aim of the study itself.

Benefits of TDD

Müller & Hagner [16] performed an experiment with students divided into two groups, test-first and traditional, with focuses on the programming efficiency, the reliability of the resultant code and program understanding. Flohr & Schneider [11] had an experiment with students divided into two groups (test-first and classical-test) for the purpose of investigating impact of test-first development process. Gupta & Jalote [17] performed an experiment with students divided in two groups (TDD and waterfall) evaluating the impact of TDD on designing, coding, and testing. Data is obtained by questionnaire and forms. Kollanus & Isomöttönen [18] performed experiment with students on understanding TDD and perception on difficulties of TDD. Data was collected by questionnaire.

Quality of produced code

George & Williams [19] had professional developers from three companies in TDD and waterfall-like control groups to investigate code quality improvements. Another controlled experiment of Janzen & Saiedian [20] examined the effects of TDD on internal software design quality. The experiment was conducted with undergraduate students in a software engineering course. Janzen et al. [21] had empirical studies in three industry short courses investigating effects of test-driven development (TDD) on internal software quality. Vu et al. [22] performed an experiment with students divided in two experimental groups (test-first and test-last) in a year-long software engineering course evaluating productivity, internal and external quality of the product, and the perception of the methodology.

Productivity improvements

Geras et al. [23] executed experiment with professional developers divided in two groups working on two problems using test-first and test-last processes to investigate productivity and software quality. Huang & Holcombe [24] had a controlled experiment with students that investigated the distinctions between the effectiveness of test-first and test-last approaches.

Quality of tests

Erdogmus et al. [25] performed an experiment with undergraduate students divided into two groups (test-first and test-last) investigating test per unit effort, quality and productivity. Madeyski [26] had an experiment with students divided in test-first and test-last groups examining branch coverage and mutation score indicator of unit tests.

Impact of experience

Müller & Höfer [27] investigated conformance to TDD of professionals and novice TDD developers. Höfer & Philipp [28] performed an experiment with professionals and students investigating if expert programmers conform to TDD to a higher extent than novice developers.

3.1.2 Test-related research

One of the key papers on software testing is: "A Survey on Testing Technique Empirical Studies: How Limited is our Knowledge" [29]. This paper provides a valuable analysis of maturity level of the knowledge on testing techniques. Several research activities with the focus on agile and testing are also identified in literature. Schooenderwoert et al. [30] are discussing different agile test techniques for embedded systems while Paige et al. [31] are creating discussion around extreme programming development for high integrity systems. Eunha et al. [32] are describing a test automation framework for agile development and testing with more focus on the developer side of testing.

3.2 Organisational perspective

A seminal document for agile development is the "Agile Manifesto" [7] explaining the main agile principles and goals behind its philosophy. This document represents a main point in our investigation on how to adopt the process while still conforming to the agile principles. By looking into some industrial reports it is possible to see how IBM is transitioning their team to agile [33], how Microsoft [34] is overcoming communication problems with testers or how the Israeli Air Force [35] is adding value to their team by introducing an outside professional tester. Some organisations are even willing to share their lessons learnt from mistakes in adopting agile [36].

3.2.1 Transitioning to Agile

We are relating our work with two approaches from the organisational perspective on how to address the role of testers issue while transitioning to agile development. Sumrell [37] reports on the experience in transitioning from Waterfall to Scrum. One of the major issues was to decide how to transform the QA team and their testing strategies to the new environment. The approach taken for the QA team is to continue to have the primary responsibility of testing, but share it with developers and project managers. Instead of testers waiting until the parts are ready for test, the new approach would be a quicker build cycle so that the QA team can do its work rather than having to wait. Retraining is needed for QA personnel to be able to instrument code for testing rather than rely on previous practices of automated testing strategies. However, unit testing becomes largely the responsibility of the developers. We can identify several characteristics of this approach. One, the role of tester is somewhat diminished because some of the testing is now done by the developer. The tester requires retraining on the technical side. The tester needs to work more closely with developers and project managers thus requiring a higher level of group working skills. We hypothesise that in such an environment, a tester needs to be given adequate training for this transition, otherwise, it is likely that he or she will fail in the new environment where they are not in control of quality, and becomes just another member of a team.

Gregory and Crispin [38] discuss in detail the role of testers in agile development. Their recommendation is to make testers a part of the development team. The role of testers is to help clarify customer requirements, turn them into tests, and help developers understand the customer requirements better. Testers need to speak the domain language of the customer and the technical language of the developers. The characteristics of this approach include an increased role for testers as the link between customers and developers in addition to their role of testing. It is a shift in their work environment as they move from the Quality Assurance Division to be part of development pairs or groups. They probably will need retraining on interpersonal skills to work closely with customers and developers more than they are used to in the past.

Chapter 4

Conclusions and Future Work

This thesis represents a set of activities conducted as part of a research process in order to identify and address potential challenges of software testing in agile development. By performing various empirical studies (questionnaire survey, literature review and controlled experiment) we brought upfront test-driven development as a noteworthy testing research direction, investigating why this practice is not utilised to a higher extent within industrial settings.

During our investigation of the current body of knowledge, we identified 18 effect areas out of which 7 are considered as limiting factors on the industrial adoption of TDD, namely, increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain- and tool-specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code.

We set up a controlled experiment with master students to investigate if developers knowledge of testing can affect adoption of TDD. Two groups of students were using TDD to solve two juxtaposing problems before and after the course on Software Verification and Validation. It is noticeable that code coverage increased in both groups after the course, but we could not identify any statistically significant difference between the groups. Further analysis of students achievements revealed lack of test cases with the focus on negative testing.

From an organisational perspective of agile adoption, we investigated possible options for transition of traditional testers into an agile environment. We

propose to define a new role for testers called “Project Mentor” which will emphasise their understanding of the complete system from a user perspective, but also utilise their technical knowledge in communication with developers.

In summary, the main contributions of this thesis are:

- The identification of TDD as a practice which is not used to the extent industry would prefer.
- Listing seven potentially limiting factors for industrial adoption of TDD
- Pointing out student’s inability to write negative test cases during controlled experiment
- Proposing the need for augmenting the TDD with the new process steps or specific testing knowledge
- Proposing the “Project Mentor” role for traditional testers in an agile environment

Concerning future work, the process of identifying limiting factors for industrial adoption of TDD was conducted using peer-reviewed scientific publications that have been addressing validity threats of their empirical study. In order to confirm significance of identified limiting factors our future work will focus on obtaining insights from industrial reports which were not covered in our previous study due to the validity requirements. This will be done in combination with industrial interviews to cover the full scope of obstacles for full utilisation of test-driven development approach.

As indicated by our study, TDD also needs to be supplemented with new process steps or test design techniques, which could potentially further enhance the robustness and the reliability of the system. In this context, we will investigate how TDD can be augmented for achieving improved code quality while keeping its fundamental principles.

In a long term research perspective, we also intent to perform an industrial case study investigating how experienced developers could benefit from testing knowledge and what kind of specific testing knowledge they need in order to increase the quality of the code artefacts they produce.

Apart from conforming the existing contributions of our research, our future work will focus on approaching as close as possible to the goal set up at the very beginning of our research:

to identify deficiencies in current testing practices in agile development environments and provide validated methods of better utilization of testers and testing techniques.

Bibliography

[1] Annual Testing Survey. Technical Report Suite 350, Quality Assurance Institute, 7575 Dr. Phillips Blvd., Orlando, FL 32819, 1994.

[2] Pankaj Jalote. *An integrated approach to software engineering (3rd Edition)*. Springer-Verlag New York, Inc., New York, NY, USA, 2005.

[3] Mikael Lindvall, Victor R. Basili, Barry W. Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie A. Williams, and Marvin V. Zelkowitz. Empirical Findings in Agile Methods. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*, pages 197–207, London, UK, 2002. Springer-Verlag.

[4] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[5] FLEXI ITEA2 Project. http://www.flexi-itea2.org/.

[6] C. Larman and V.R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47 – 56, june 2003.

[7] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development. http://www.agilemanifesto.org/, 2001.

[8] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors. *Structured programming*. Academic Press Ltd., London, UK, 1972.

[9] Ian Sommerville. *Software engineering (6th ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[10] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[11] Thomas Flohr and Thorsten Schneider. Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings. In Jrgen Münch and Matias Vierimaa, editors, *Product-Focused Software Process Improvement*, volume 4034 of *Lecture Notes in Computer Science*, pages 305–318. Springer Berlin / Heidelberg, 2006.

[12] Barry Boehm and Richard Turner. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software*, 22:30–39, 2005.

[13] Adnan Causevic, Iva Krasteva, Rikard Land, A. S. M. Sajeev, and Daniel Sundmark. An Industrial Survey on Software Process Practices, Preferences and Methods. (ISSN 1404-3041 ISRN MDH-MRTC-233/2009-1-SE), March 2009.

[14] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833 – 859, 2008.

[15] Laurie Williams. Agile Software Development Methodologies and Practices. *Advances in Computers*, 80:1–44, 2010.

[16] M.M. Müller and O. Hagner. Experiment about test-first programming. *Software, IEE Proceedings* -, 149(5):131 – 136, October 2002.

[17] Atul Gupta and Pankaj Jalote. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 285–294, Washington, DC, USA, 2007. IEEE Computer Society.

[18] Sami Kollanus and Ville Isomöttönen. Understanding TDD in academic environment: experiences from two experiments. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 25–31, New York, NY, USA, 2008. ACM.

[19] Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337 – 342, 2003.

[20] David S. Janzen and Hossein Saiedian. On the Influence of Test-Driven Development on Software Design. *Software Engineering Education and Training, Conference on*, pages 141–148, 2006.

[21] David S. Janzen, Clark S. Turner, and Hossein Saiedian. Empirical software engineering in industry short courses. Software Engineering Education Conference, Proceedings, pages 89–96, 2007.

[22] John Huan Vu, Niklas Frojd, Clay Shenkel-Therolf, and David S. Janzen. Evaluating Test-Driven Development in an Industry-Sponsored Capstone Project. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 229–234, Washington, DC, USA, 2009. IEEE Computer Society.

[23] A. Geras, M. Smith, and J. Miller. A Prototype Empirical Evaluation of Test Driven Development. In *Proceedings of the Software Metrics, 10th International Symposium*, pages 405–416, Washington, DC, USA, 2004. IEEE Computer Society.

[24] Liang Huang and Mike Holcombe. Empirical investigation towards the effectiveness of Test First programming. *Inf. Softw. Technol.*, 51:182–194, January 2009.

[25] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31:226–237, 2005.

[26] Lech Madeyski. The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment. *Inf. Softw. Technol.*, 52:169–184, February 2010.

[27] Matthias Müller and Andreas Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12:593–615, 2007.

[28] Andreas Höfer and Marc Philipp. An Empirical Study on the TDD Conformance of Novice and Expert Pair Programmers. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski,

Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 33–42. Springer Berlin Heidelberg, 2009.

[29] N. Juristo, A. M. Moreno, and S. Vegas. A Survey on Testing Technique Empirical Studies: How Limited is our Knowledge. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, pages 161–, Washington, DC, USA, 2002. IEEE Computer Society.

[30] Nancy Van Schooenderwoert and Ron Morsicato. Taming the Embedded Tiger - Agile Test Techniques for Embedded Software. In *Proceedings of the Agile Development Conference*, pages 120–126, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Richard F. Paige, Howard Chivers, John A. McDermid, and Zoë R. Stephenson. High-integrity extreme programming. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1518–1523, New York, NY, USA, 2005. ACM.

[32] Eunha Kim, Jongchae Na, and Seokmoon Ryoo. Developing a Test Automation Framework for Agile Development and Testing. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 8–12. Springer Berlin Heidelberg, 2009.

[33] Susan D. Shaye. Transitioning a Team to Agile Test Methods. In *Proceedings of the Agile 2008*, pages 470–477, Washington, DC, USA, 2008. IEEE Computer Society.

[34] Michael Puleio. How Not to Do Agile Testing. In *AGILE '06: Proceedings of the conference on AGILE 2006*, pages 305–314, Washington, DC, USA, 2006. IEEE Computer Society.

[35] David Talby, Orit Hazzan, Yael Dubinsky, and Arie Keren. Agile Software Testing in a Large-Scale Project. *IEEE Softw.*, 23:30–37, July 2006.

[36] Kay Johansen and Anthony Perkins. Establishing an Agile Testing Team: Our Four Favorite "Mistakes". In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile*

Methods - XP/Agile Universe 2002, pages 52–59, London, UK, 2002. Springer-Verlag.

[37] Megan Sumrell. From Waterfall to Agile - How does a QA Team Transition? In *AGILE '07: Proceedings of the AGILE 2007*, pages 291–295, Washington, DC, USA, 2007. IEEE Computer Society.

[38] Lisa Crispin and Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009.

II

Included Papers

Chapter 5

Paper A:
An Industrial Survey on Contemporary Aspects of Software Testing

Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat
In proceedings of the International Conference on Software Testing (ICST), Paris, France, April 2010

Abstract

Software testing is a major source of expense in software projects and a proper testing process is a critical ingredient in the cost-efficient development of high-quality software. Contemporary aspects, such as the introduction of a more lightweight process, trends towards distributed development, and the rapid increase of software in embedded and safety-critical systems, challenge the testing process in unexpected manners. To our knowledge, there are very few studies focusing on these aspects in relation to testing as perceived by different contributors in the software development process.

This paper qualitatively and quantitatively analyses data from an industrial questionnaire survey, with a focus on current practices and preferences on contemporary aspects of software testing. Specifically, the analysis focuses on perceptions of the software testing process in different categories of respondents. Categorization of respondents is based on safety-criticality, agility, distribution of development, and application domain. While confirming some of the commonly acknowledged facts, our findings also reveal notable discrepancies between preferred and actual testing practices. We believe continued research efforts are essential to provide guidelines in the adaptation of the testing process to take care of these discrepancies, thus improving the quality and efficiency of the software development.

5.1 Introduction

Software testing, as a practice, has been able to successfully evolve over time and provide efficient and constant support for improvements in software quality. On the other hand, testing is still notorious for its massive resource consumption within software projects. To this date, much of the research efforts on software testing have been focusing on designing new techniques, as well as investigating their effectiveness in real development contexts. However, during the entire history of software development, testing methods and techniques have struggled to keep up with the ever faster evolution and trends in software development paradigms. We cannot expect any favourable change in this state of affairs, unless a conscious effort is made in anticipating the trends, learning the stakeholder mindsets, and pinpointing the problem areas. It is our belief that such an effort could help in efficiently allocating the testing resources toward a specific context or in proactively deciding the testing research agenda in general. To our knowledge, there exist no detailed investigations with such a perspective. The research presented in this paper is a small step in this direction, in that it specially focuses on the stakeholder perspectives on some contemporary aspects related to testing. The specific research question we address in this paper is: *Is it possible to identify and list main discrepancies between current and preferred testing practices that could be considered as obstacles for software testing practitioners?*

By qualitatively and quantitatively analysing the results of a recent questionnaire survey on practices and preferences in industrial software development, with respect to the above research question, we have identified a number of areas and practices where the preferred practice significantly differs from what is perceived as the actual current practice. We believe that these areas and practices assist in pointing out directions for future research within software testing.

The contributions of this paper are three-fold:

- A qualitative analysis on practices and preferences in testing of different contemporary categories of software development professionals (Section 5.3).
- A qualitative analysis on techniques and tools used in contemporary testing (Section 5.4).
- A quantitative analysis of satisfaction with current testing practice among different categories of respondents (Section 5.5).

5.2 Research Method

We base our analyses in this paper on data from a recent industrial survey on software development practices and preferences. This survey was a combined effort of several researchers with diverse research foci, all integrated to one study to minimise the responders time. The survey was performed using a web-based questionnaire, and the invitation was distributed among industrial software development companies using, e.g., the FLEXI and NESSI European project networks. More information about the questionnaire, as well as all data, is available as a technical report [1]. The questions pertinent for our research were embedded in the larger set of questions and were formulated in such a way as to provide the list of discrepancies indirectly rather than asking the questions in a direct way which could be either provocative or could result in a no response since some of the respondents may not want to present themselves as opinionated.

5.2.1 Categorization of Respondents

In our analysis, we categorize survey respondents according to five aspects of contemporary software development, namely:

1. Agility of Development Process
2. Distribution of Development
3. Domain of Product(s)
4. Safety-Criticality of Product(s)
5. Amount of Testing performed by Respondent

In order to categorize respondents according to these aspects, we make use of a set of categorizing questions. For example, a respondent belongs in the "Agile" category of respondents if the answer to the question "*Our current software development process is:*" is "*Agile*".

Note that any respondent may be included in several non-mutually exclusive categories. For example, a respondent might be categorized as a *tester*, working with *safety-critical* software in a *non-distributed* development (examples of mutually exclusive categories are *testers* and *non-testers*, and *distributed* and *non-distributed*). The domain categories are not mutually exclusive, as some respondent companies develop software for multiple domains.

This cross-coupling between categories was an intentional outcome of our research objective of unearthing more inter-related answers from the respondents indirectly.

The categorization of respondents and the categorizing questions can be viewed in Table 5.1.

Categorizing Criterion	Categorizing question and response		Category
	Question	Response	
Agility of Development Process	Our current software development process is:	"Agile"	Agile
		Other	Non-agile
Distribution of Development	In our team: all of the team members are collocated in one building	Yes	Distributed
		No	Non-distributed
Safety-Criticality of Product(s)	If the software developed in our current project fails, the maximum damage could be the loss of:	"Many lives" or "A single life"	Safety-critical
		Other	Non-safety-critical
Amount of testing performed by Respondent	At work, I perform the following activities [indicate how often on a scale of 0 to 7]: [testing]	1-7	Testers
		0	Non-testers
Domain of Product(s)	The software we build in our project is:	Web Software	Web
		Desktop Software	Desktop
		Embedded software	Embedded

Table 5.1: Categorization of Respondents

5.2.2 Question Selection

Since the amount of questions in the survey data is quite large (a total of 260 questions were included in the questionnaire), we wanted to focus only on questions that explicitly or implicitly related to the testing process. For the explicitly test-related questions, this process was trivial, but for the more implicit

questions, the process was subjective. Our rationale was that, for a question to be test-related, the practice queried on needs to directly affect the testing process. An example of such an implicitly test-related question is the question "*Management should encourage regular interaction between developers and customers/business people*", since regular interaction between developers and customers directly may affect how acceptance testing is performed.

Moreover, for each contemporary aspect, we selected and studied a subset of test-related questions that specifically applied to that particular contemporary aspect. For example, the question "*We never have to wait for source code in order to start the testing process*" is highly interesting in a distributed development context, where communication between different development teams may be impaired by, e.g., geographical and cultural distances within the organisation.

5.2.3 Scales Used for Answers

Respondents were providing their opinion by selecting one of the options from a given scale of answers against each question. Two different scales (with 7 or 8 divisions) were used in our survey depending on the type of question. The 7-scale options represent: "Very strongly disagree", "Strongly disagree", "Disagree", "Neither agree nor disagree", "Agree", "Strongly agree"and "Very strongly agree". For certain analyses, 7-scale answers were mapped to numerical values in the (-3, 3) interval. This would mean that a "-2" value should be interpreted as the respondent strongly disagrees with the statement provided in the question, whereas a "1" should be interpreted as the respondent agrees. The 8-scale answers were used for questions in which respondent were asked to provide the level of usage for some testing types. This way they could choose an option from 0 to 7 where 0 means they never use it and 7 they always use it.

In Section 5.3, Table 5.3 to Table 5.7 present the data from the questions where the used scale is implicitly assumed from left to right as the headings of the columns with numeric values. The numerical values in turn represent the number of respondents in that particular group who have answered that particular question with the same answer option.

5.3 Testing Practices and Preferences

For a better understanding of our targeted respondents group, a general overview of respondent demographics for the complete survey is presented in Table 5.2.

Gender	Male	73
	Female	10
Age	25-29	13
	30-34	22
	35-39	25
	40-49	14
	50-59	6
Education qualification	Udergraduate or lower	3
	Bachelor degree	21
	Postgraduate degree	45
	PhD or above	14
IT work experience	1-4 years	16
	5-8 years	22
	9-12 years	18
	13-16 years	14
	More than 16 years	13
Team size	I am not in a team	6
	1-5 people	25
	6-10 people	22
	11-15 people	10
	16-20 people	6
	21-50 people	9
	More than 50 people	4
End product of project is	a software part/component which is to be integrated	15
	a software service	4
	a software system that will be used by end users	39
	other	5

Table 5.2: Respondent Demographics

Now we analyze data regarding practices and preferences in testing in the different categories of respondents. Please note that some questions are conditional based on earlier responses which as a result provide less number of responders compared to the complete survey. This analysis is performed on five different aspects where each aspect is individually presented within its appropriate subsection.

5.3.1 Agile vs. Non-Agile

In the remainder of this section, we will use the term “agile respondents” to refer to the group of respondents who claim that agile is their current development process. Similarly, we will use the term “non-agile respondents” to refer

to the group of respondents that claim to use any other development process (e.g., waterfall, adaptive, ad-hoc, etc.).

Regular interaction with customers is a central theme in agile development [2] and it can as a benefit provide a continuous assistance in creating and validating acceptance tests. However, looking at the responses from the agile respondents (see Table 5.3), there are (clear) indications that this agile practice is not followed to the extent they would prefer. To some extent, customer interaction is limited to elicitation of requirement and acceptance testing. This is true also for the non-agile respondents, but they are, to a larger extent, happy with the current practice.

The agile respondents are not averse to changes, especially compared to the non-agile respondents, who have a slight tendency to discourage customers from changing requirements. Moreover, in the preferred practice, the agile respondents would like to be even less restrictive to change, whereas the non-agile respondents would like to discourage customers in changing requirements even more.

Changing working software is, as expected, favoured and even more preferred among the agile respondents. Surprisingly, the non-agile respondents do not mind such changes, even though they are less enthusiastic than the agile respondents.

Changes to working software with the specific purpose of improving the structure of the code (i.e., refactoring) is highly preferred among both groups, but the agile respondents appear to be following this practice to a significantly higher extent (albeit not quite at the desired level). It could probably be claimed that the high acceptance of changes to working code is a result of confidence provided by the regression suites created by a test-driven development practice [3]. However, this is not a conclusion we can draw solely based on our data.

Test driven development (TDD) is one of the most prominent practices used in agile development. However, our respondents, especially the ones working in agile processes, are leaning towards disagreement that this practice is in place at their current organisation. Both groups are agreeing their preference is to use this practice to a higher extent.

As a summary, we can observe two facts from this data:

1. Respondents working in agile processes are not happy with current test first practice (possibly because they are in an early phase of adoption)

2. Non-agile respondents are unknowingly following agile testing practices.

Interaction with customers/business people should be for capturing the requirements at the beginning of the project and then for acceptance testing at the end of the project								
Current Practice	Agile	1	0	4	1	2	1	2
	Non-Agile	1	5	3	10	12	4	1
My preference	Agile	3	2	2	0	1	1	2
	Non-Agile	4	5	4	4	6	11	2
Customers/ business people should be discouraged from changing requirements once they are specified								
Current Practice	Agile	4	0	3	2	1	1	1
	Non-Agile	1	4	6	10	10	3	1
My preference	Agile	4	2	3	2	0	1	0
	Non-Agile	1	2	9	5	9	8	2
Once a piece of code starts working, it should rarely be modified								
Current Practice	Agile	3	1	3	2	1	0	0
	Non-Agile	2	4	6	13	6	4	1
My preference	Agile	3	2	4	1	0	0	0
	Non-Agile	0	9	8	7	4	6	1
Regular changes to working code should be encouraged if they improve the code in some way (e.g. its design, its structure etc.)								
Current Practice	Agile	0	0	0	2	4	3	1
	Non-Agile	1	4	6	13	8	2	0
My preference	Agile	0	0	0	0	2	6	2
	Non-Agile	0	1	5	8	11	7	3
Test cases should be written before writing code								
Current Practice	Agile	1	2	4	1	2	0	0
	Non-Agile	1	7	11	8	5	3	0
My preference	Agile	1	1	1	0	3	4	0
	Non-Agile	2	0	5	6	9	12	1

Table 5.3: Survey data for Agile vs. Non-Agile

5.3.2 Distributed vs. Non-distributed

By answering the question if all of the team members are collocated in one building, responders are grouped into distributed and non-distributed categories. Questions were asked only to responders who claim to have some testing activities at their work, i.e., the distributed and the non-distributed respondents make up disjoint subsets of the testers category. Collected data is presented in Table 5.4.

Please indicate how strongly you agree or disagree with the following statements with respect to your testing experience in current organisation								
We never have to wait for source code in order to start the testing process	Distributed	3	2	4	1	3	3	2
	Non distributed	0	3	4	7	2	0	2
The necessary infrastructure for executing test cases is always in place	Distributed	0	2	7	0	6	2	1
	Non distributed	0	5	7	3	1	0	2
There are no changes done on code during integration testing	Distributed	2	5	3	0	8	0	0
	Non distributed	1	3	10	2	0	1	1
During integration testing, I do not mind code to be changed while I am testing it	Distributed	2	1	9	1	5	0	0
	Non distributed	1	0	4	6	6	1	0

Table 5.4: Survey data for Distributed Development

When it comes to not having source code available on time to start the testing process, testers working in distributed environments give very diverse answers. In average, they can be considered the same, as most of non-distributed responders, who claimed they neither agree nor disagree this problem exist in their organisation. Since we expected testers working in distributed environments to experience this problem in higher extend, we can only claim that working in distributed development is not a main cause of delays in source code delivery within our respondents.

Not having a proper infrastructure for testing in place could potentially slowdown the testing process. We expected this in particular to be a problem for distributed development, and even though answers were again diverse they were very close to neither agree nor disagree that problem with missing infrastructure exists. However, non-distributed respondents to a slightly higher

extent indicated to experience the problem of not having necessary infrastructure in place. Again we could claim within our responders that the problem with not having a proper infrastructure for testing is not directly related to using distributed development process.

Changes done on code could produce further delays and introduce complexity during integration testing. One would naturally expect this to be in much higher degree present in distributed development rather than non-distributed. Our data shows this is not entirely true for our respondents. Both groups tend to disagree there are no changes on code during integration testing with slightly higher level of disagreement within non-distributed respondents. On the other hand, respondents working in distributed development prefer to disagree that they do not mind code to be changed while they are testing it during integration. Non-distributed respondents preferences point out to neither agreement nor disagreement with this statement.

We expected to recognize potential challenges with testing related to distributed development by analysing our data, but as presented in this section, we did not find any such problems among our respondents.

5.3.3 Domain

In the domain categorization, respondents were categorized according to answers to the following question: “The software we build in our project is:”. Possible answers were “Desktop”, “Web” or “Embedded”. In contrast to the other respondent categorizations in this paper, the domain categories are non-mutually exclusive. Hence, if respondents are working in companies developing products in several domains, they are included in all these domain categories. Survey data for the domain categorization is presented in Table 5.5.

In the domain categorization, our expectation was to find that a lack of testing infrastructure would cause problems for testing particularly in the embedded system domain, mainly due to problems of hardware/software co-design and hardware availability in early stages of development. We further believed that this would negatively affect the time available for testing. Moreover, as commonly stated (e.g., in [4]), we hypothesized that development of web and desktop software would, to a larger extent, be influenced by lightweight and agile methods, whereas development of industrial and embedded software would typically follow a more traditional, plan-driven process. There is, however, no support for the latter assumption in the data, as development method seems to be independent of software domain among our respondents.

We do note a more prominent lack of availability of a proper testing infrastructure in the embedded system domain, but not fully to the extent we might

Please indicate how strongly you agree or disagree with the following statements with respect to your testing experience in current organization:									
The necessary infrastructure for executing test cases is always in place	Desktop	0	2	4	2	0	2	2	
	Web	0	2	4	0	5	2	0	
	Embedded	0	4	6	1	2	2	0	
I have enough time to test the software before its deployment	Desktop	1	3	0	3	3	2	0	
	Web	0	1	5	1	4	2	0	
	Embedded	0	2	5	3	2	3	0	
In our project we use the following testing types									
Unit testing	Desktop	1	1	0	0	0	2	0	1
	Web	0	1	0	1	0	2	3	4
	Embedded	1	2	1	2	1	3	0	2
Functional black-box testing of the whole system	Desktop	1	0	1	0	2	0	1	1
	Web	2	0	0	1	3	0	4	1
	Embedded	1	0	0	0	3	1	2	5
Performance testing (including load and stress testing)	Desktop	1	0	2	0	0	0	1	2
	Web	2	1	0	0	1	1	2	4
	Embedded	1	0	1	1	2	2	3	2
In my opinion, the ideal level for each of the following testing types in our project should be									
Unit testing	Desktop	0	1	0	0	0	1	0	4
	Web	0	1	0	0	1	1	2	6
	Embedded	1	2	1	1	0	1	3	3
Functional black-box testing of the whole system	Desktop	1	0	0	0	0	2	1	2
	Web	1	0	0	0	0	4	2	4
	Embedded	1	0	0	0	0	2	2	7
Performance testing (including load and stress testing)	Desktop	1	0	1	0	0	2	0	2
	Web	1	0	0	1	1	3	2	3
	Embedded	1	0	0	0	3	2	0	6

Table 5.5: Survey data for Application Domain

have expected. Furthermore, it is worth mentioning that this deficiency does not seem to significantly affect the experienced sufficiency of time available for testing, which is quite equal between the domains.

Regarding the importance of different testing types, in the current practice, web development seems to put a large emphasis on unit testing, whereas embedded system development to a higher degree focuses on functional system testing. In the desktop system development category of respondents, there is no clear indication of a coherent current practice.

Comparing the current practice with what is considered the ideal practice, there are a few noteworthy discrepancies. Generally, among all categories of respondents, there is a preference towards more rigorous testing at all levels, particularly visible in the functional system-level testing. Notable is also the degree in which embedded system developers would like to increase load and stress testing, a practice where they feel that the current level of practice is insufficient.

5.3.4 Safety-criticality

Prior to the analysis of respondent data, we expected safety-critical respondents to lean towards more traditional types of development and testing in the current practice, but we were curious and more hesitant regarding their preferred practice.

When it comes to customer involvement, there is a significant difference between the safety-critical respondents, and the non-safety-critical respondents (see Table 5.6). The safety-critical respondents to a large extent limit customer interaction to requirements elicitation in the beginning of the project, and acceptance testing at the end of the project. This is something that cannot be seen in the group of non-safety-critical respondents, where the current practice varies. Interestingly, the safety critical respondent preference is to further decrease customer involvement, whereas most non-safety critical respondents would prefer an increase.

Discouraging customers/business people to change requirements is a practice which both groups neither agree nor disagree to currently exists in their organisation. We expected safety-critical environment to be more resistant to change than non-safety-critical respondents, but within our respondents this is not a case. Interestingly, the preference of both groups does not change significantly from the current practice. We could say that changing requirements is not something our respondents would agree easily with, but it is rather something they must accept, regardless of software criticality.

Both the safety-critical and the non-safety-critical categories of respondents state that writing test cases before writing code is mostly not considered as the current practice. However, while the non-safety-critical respondents seem quite willing to change this situation, the safety-critical respondents show no interest as a group in changing towards a more test-driven development. This is noteworthy considering the fact that empirical studies seem to ascribe test-driven developed code a high external code quality [5] [6] [7]. For fairness sake, it is not trivial to see how such a practice would affect, and be affected by, other specific aspects of safety-critical system development, e.g., fulfilment of safety certification standards.

Interaction with customers/business people should be for capturing the requirements at the beginning of the project and then for acceptance testing at the end of the project								
Current practice	Safety-critical	0	0	0	2	2	3	1
	Non-safety-critical	2	4	7	8	12	2	2
My preference	Safety-critical	0	1	0	0	0	5	2
	Non-safety-critical	7	5	6	3	7	7	2
Customers/ business people should be discouraged from changing requirements once they are specified								
Current practice	Safety-critical	2	0	2	2	2	0	0
	Non-safety-critical	3	3	7	10	9	4	1
My preference	Safety-critical	3	0	3	0	1	2	0
	Non-safety-critical	3	3	9	7	8	7	1
Test cases should be written before writing code								
Current practice	Safety-critical	1	1	4	1	1	0	0
	Non-safety-critical	1	8	9	8	6	3	0
My preference	Safety-critical	1	0	3	2	1	1	0
	Non-safety-critical	2	1	2	3	11	15	1

Table 5.6: Survey data for Safety-Criticality

5.3.5 Testers vs. Non-Testers

Programming should start only after the design is completed								
Current practice	Testers	2	6	10	9	10	1	0
	Non-Testers	1	2	1	2	2	1	0
My preference	Testers	4	5	6	7	5	9	2
	Non-Testers	1	0	1	1	2	4	0
The main focus of the team should be to get the code to work								
Current practice	Testers	0	1	3	6	11	13	4
	Non-Testers	0	2	1	1	3	0	1
My preference	Testers	0	2	6	5	6	13	6
	Non-Testers	0	2	1	2	2	2	0
The main focus of the team should be on the production of all artefacts (e.g. design documents, requirements documents) not just code								
Current practice	Testers	1	3	5	9	16	4	0
	Non-Testers	0	0	3	3	3	0	0
My preference	Testers	1	2	3	7	10	12	3
	Non-Testers	0	0	1	1	1	4	2
Once a piece of code starts working, it should rarely be modified								
Current practice	Testers	5	4	7	13	3	4	0
	Non-Testers	0	1	2	2	4	0	0
My preference	Testers	3	9	11	7	2	3	1
	Non-Testers	0	2	1	1	2	3	0

Table 5.7: Survey data for Testers vs. Non-testers

Based on responders' answers to question on how often they perform testing activities, we grouped their response into two categories: Testers and Non-testers. We expected to have insights into differences on how testing related

activities are seen from these roles. Data is presented in Table 5.7.

Testers and non-testers opinions whether programming should start only after the design is completed, do not differ significantly. Both groups are stating this is, to some extent, present in their current practice but importantly to notice, both groups equally point out preference on having this practice in place. Similar results can be seen in question if the main focus of the team should be on the production of all artefacts and not just code. Neither testers nor non-testers agree or disagree this practice exist in their current organisations. However, both group preferences show high level of agreement that team focus should be tailored towards generating all artefacts of software development.

Question if the main focus of the team should be to get the code to work show us some difference in group opinions. Testers think that this philosophy is slightly present in their current organisation, while non-testers tend to disagree. Both groups seem satisfied with their current practice since preference on this idea does not change. Another philosophy was investigated by a question on if once a piece of code starts working, it should rarely be modified. Here, testers show tendency to disagree that this approach exist in their current organisation, while non-testers neither agree nor disagree with it. Interestingly, testers' preferences are to further disagree with this practice, while non-testers show some level of agreement this practice should exist in organisations.

5.4 Techniques and Tools

Respondents, who previously stated to perform testing activities at work, were additionally asked questions regarding tools and techniques currently in use within their organisation. Those questions did not have any predefined answers since we expected the respondents to provide us with inside information about existing testing practices. We expect that an appropriate grouping of information (for e.g.. domain-wise) could be beneficial for new practitioners. Table 5.8 and Table 5.9 present the respondents answers followed by our qualitative analysis of the data.

Based on the provided responses (shown in Table 5.8) we notice, as expected, that it is very common for organisations to have defined levels of testing. Those levels are usually unit, integration and system level testing. Besides having testers, testing activities are also performed by developers. In most cases, unit level testing using a white box approach is a responsibility of developers, whereas integration and system level testing are done by dedicated testers performed mostly as black box testing. Interestingly, we have not found a big presence of automation effort in testing.

Which testing technique do you use in your organisation? (if you are not sure of the name of the technique, try to explain in short how you perform testing)
Unit testing, integration testing and functional testing.
Low level: Lint, Code coverage, Manual code review. High level: Integration test, Regression test (to verify legacy functionality), Function test (verify new stuff), System test (from an end user perspective)
White box unit testing
unit testing w test coverage strategies (all statements, black-box behaviour) system testing with real hardware done by or together with our customer
No automated testing, only interactive. Though, we have tools which measures coverage as well as performance bottlenecks.
unit testing by developers, per use case manual testing done by testers
No technique as we just do prototypes. We dont test it towards test documents, since just on user tests.
Module test, integration test on dedicated hardware, test on complete machine
1. Manually develop and debug using whatever equipment available using PC/Windows or actual target hardware. 2. Module testing on workstation platform using PC/Windows in a repetitive form. 3. Manual repetitive integration tests on actual target hardware platform. 4. Automated tests on actual hardware and/or system. (not all tests use have automated test cases)
White-box (developers testing their code using debuggers) Black-box (on system level) Unit testing (not so common) Automatic testing (no so common)
Limited regression testing, limited automated user interface testing
It varies from project to project. My current project writes unit tests at the same time as the code, and different people do system testing.
Ad hoc testing, i.e. testing only specific functionality rather than full regression testing for each release.
Testframe
Vast test automation, explorative testing, black-box - white-box, etc, etc...
manual testing
Unit Testing and Integration Testing (White box) by the development team. Black Box (System Testing and Performance Testing) by an independent Test Team
Unit tests, integration & regression tests.
Use case testing State transition testing Classification tree method Boundary value analysis
Ad hoc
Different in different projects.
Acceptance testing against functional and non-functional requirements, creating system test automation in business value order. Unit testing and module testing done by developers.
User-story approach, old-fashioned test-case approach, exploratory testing, test automation, TDD
Component testing done by designers during development. Function testing before delivery to integration branch. Integration testing. System/load testing.

Table 5.8: Respondent Answers on Techniques in use

With respect to tools used as support for testing (shown in Table 5.9), from respondent answers we can notice that there is utilization of both open source and proprietary tools. However, open source testing tools seem to be mostly used for unit testing whereas for higher level of testing, a proprietary tool is in place. Even a few in-house developed testing tools are stated in the answers.

Do you use any tools for testing within your organisation? Please provide us with their names:
Not any specific tool.
Expect/TCL, Jcat (java based tool), Perl. Also some proprietary testing platforms based on previously named tools.
NUnit
internally developed tool, simple script on top of a CAN & I/O simulator and same scripts using real CAN & I/O when software is downloaded on target.
Two tools named something with ”coverage” and ”perform” (cannot remember the company behind)
JUnit, JEmitter
PC-Lint and/or Programming Research QA C for static analysis, MSDevStudio for code coverage analysis of test cases, homebrewed testing harness suited for the current development tools, RTOS supported functions for timing analysis. JTAG/BDM-debuggers for on-target testing. National Instruments LabView using automated test cases derived from requirement tools.
TestComplete (tool for automatic tests)
Yes. Do not know.
Again it varies from project to project. I think my current project uses JUnit, but I have no time to get involved.
Not personally.
code coverage, JUnit(UnitTest)
change control, bug tracking, test case management, etc., etc.
Proprietary in most cases.
Check, valgrind
tcl/expect, Test-RT
Test Director CTE Test execution tools (self-made, using Labview and Perl) NUnit for developer tests
No
Our proprietary test automation system
In-house test automation
TTCN. Load/traffic generators (not sure of name, SipP?).

Table 5.9: Respondent Answers on Tools in use

5.5 Satisfaction of Current Practice

Our third and final analysis in this paper is a quantitative analysis on the satisfaction of current testing practices. If one agrees that knowledge of required process improvements is, to some extent, intrinsic within software development organisations, this information may provide valuable guidelines for future research directions in this area. Below, we will discuss *dissatisfaction* of current practice rather than *satisfaction*. This might seem overly negative, but is rather an effect of measuring the absolute value of the difference between the perceived current and the preferred practices. Hence, a high value indicates a high degree of dissatisfaction. Needless to say, a low degree of dissatisfaction equals a high degree of satisfaction.

Here, individual dissatisfaction of current testing practices is defined as the mean absolute difference between the perceived current practice, and the preferred practice for all testing-related questions for a single respondent. Dissatisfaction among a category of respondents is defined as the mean of all individual dissatisfactions for the respondents within that category. Formally, given the set R of respondents in a particular category, and the set Q of test-related questions, the dissatisfaction $d_{Q,R}$ is defined by

$$d_{Q,R} = \frac{1}{|Q||R|} \sum_{q \in Q} \sum_{r \in R} |c_{q,r} - p_{q,r}|$$

where, $c_{q,r}$ and $p_{q,r}$ refer to the current and the preferred practice of question q as reported by the respondent r.

5.5.1 Satisfaction within Different Categories of Respondents

For the categorization used in Section 5.3, the dissatisfaction results are shown in Table 5.10.

As can be seen in the table, the differences between the categories are not remarkably large. The difference between the most dissatisfied category of respondents (non-distributed) and the most satisfied category of respondents (safety-critical) is 0.359, corresponding to little over one third of a grade per question and respondent on a 7-grade scale. Nevertheless, the safety-critical respondents stand out as the most satisfied categories of respondents. This can possibly be attributed to the fact that the training levels in such organisations are very high and focused which make them fully believe in and be confident of the activities and practices. Further investigation will be necessary to confirm that such a conclusion is justifiable from a technological point of view. It

Respondent category	*Dissatisfaction*
Safety-critical respondents	0.660
Agile respondents	0.728
Desktop respondents	0.821
Embedded respondents	0.875
Distributed respondents	0.880
Web-based respondents	0.910
Testers	0.931
Non-testers	0.933
Non-safety-critical respondents	0.983
Non-agile respondents	0.995
Non-distributed respondents	1.019

Table 5.10: Dissatisfaction within categories of respondents

is also worth mentioning that the satisfaction of current testing practice among respondents doing distributed development is actually higher than that of respondents doing non-distributed development. The obvious assumption would be that distributed development puts additional strains on the efficiency of testing. A possible explanation of these results might be that a distribution of development enforces the software development organization to be more conscious about its testing strategy.

5.5.2 Satisfaction with Particular Testing Practices

In order to analyse the dissatisfaction with regards to a particular practice, denoted by $d_{q,R}$, we make use of a special case of the above equation, where q is the question on the practice of interest and $Q = \{q\}$.

$$d_{q,R} = \frac{1}{|R|} \sum_{r \in R} |c_{q,r} - p_{q,r}|$$

It should also be noted that a high dissatisfaction in a question only tells us that there is a large difference between the current and the preferred practice. It does not explicitly tell us the nature of this dissatisfaction, nor does it mean that respondents agree on whether the practice queried on should be increased or decreased. A prime example of this is the question "*Once a piece of code starts working, it should rarely be modified*", which has a dissatisfaction of 0.935, which is quite high in relation to the total data set. However, taking into

account whether the respondents want an increase or a decrease of the practice (mathematically, this is equivalent to disregarding the absolute value operation on the difference between current and preferred practice), we end up with a value of 0.109, indicating that there is almost an equal desire to increase the practice as it is to decrease it, among the set of respondents.

Question	*Dissatisfaction*
Changing working code should not be encouraged but cannot be prevented	0.500
Testing should be a defined phase in project development	0.587
Procedures and processes should be allowed to be changed often if the change brings in an improvement	0.652
Project planning should be incremental, one iteration at a time	0.681
How much functionality is in the current working code should be the sole criteria for determining progress of the project	0.696

Table 5.11: Questions with the Lowest Degree of Dissatisfaction

In Table 5.11, the five questions with the lowest degree of dissatisfaction are presented. Notably, two out of the three questions with the lowest degree of dissatisfaction concern changes to working code for the purpose of quality improvement or customer satisfaction. Considering the fact that these statements are mostly agreed to by the respondents, it seems to be generally accepted, both in theory and in practice, that changes are inevitable in the software development process, even though they may pose difficulties.

Table 5.12 displays the five statements with the highest degree of dissatisfaction. Out of all the queried practices, test-driven development seems to be the practice where the difference between the preferred practice and the current practice is most significant. Further analysis of the questionnaire data reveals that the cause of this dissatisfaction is that test-driven development is not practically used to the extent that is desired by the respondents. As already shown in Section 5.3, this is true both for agile- and non-agile respondents.

The question "*Programming should start only after the design is completed*", is notable not only because of its high degree of dissatisfaction, but also that because it polarized respondents to such an extent. Seven respondents recognized this as the current practice, and would like a decrease. On the other hand, twenty respondents felt that the programming starts too soon and should

Question	*Dissatisfaction*
Test cases should be written before writing code	1.609
Programming should start only after the design is completed	1.271
Comprehensive documentation should be an essential part of software development	1.250
There should be general guidelines and principles for software development but not detailed rules	1.204
Management should encourage regular interaction between developers and customers/business people	1.200

Table 5.12: Questions with the Highest Degree of Dissatisfaction

be postponed until the design is completed.

For the remaining three questions in the table, the dissatisfaction in the practices is due to the fact that the practices are preferred, but not followed.

5.6 Conclusion

In this paper we present our analysis results from an industrial questionnaire survey on the current software development practices and preferences, specifically in relation to testing. The survey has unique features such as strategic embedding of multi-purpose questions and categorisation of respondents on contemporary aspects which enable us to gain qualitative insights. The survey in addition to confirming some popular beliefs also lists several noteworthy findings from the perspectives of respondent categories such as safety-criticality, agility, distribution of development, and application domain. These findings clearly depict negative discrepancies between the current practices and the perceptions of the respondents, thus meeting our research objective presented in the introduction.

One of the noteworthy testing research directions from an industrial perspective seems to be test driven development as indicated by the results of the survey. Our ongoing research work attempts to refine these findings through directed interviews as well as through further investigations in a wider context. We are also working on the definition of a methodology for dynamically incorporating such findings in the management decisions on strategic challenges such as introduction of new technologies, processes, and effort allocations in relation to testing.

5.7 Acknowledgments

This work was supported by the VINNOVA through the ITEA2 project FLEXI. The authors want to express their thanks to all the questionnaire respondents and the people involved in earlier phases of this research.

Bibliography

[1] Adnan Causevic, Iva Krasteva, Rikard Land, A. S. M. Sajeev, and Daniel Sundmark. An Industrial Survey on Software Process Practices, Preferences and Methods. (ISSN 1404-3041 ISRN MDH-MRTC-233/2009-1-SE), March 2009.

[2] Geir Kjetil Hanssen. Agile Customer Engagement: a Longitudinal Qualitative Case Study. In *In Proceedings of International Symposium on Empirical Software Engineering (ISESE) (Rio de*, 2006.

[3] Kent Beck. *Test Driven Development*. Addison Wesley, November 2002.

[4] Jussi Ronkainen and Pekka Abrahamsson. Software development under stringent hardware constraints: Do agile methods have a chance. In *Proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering 2003*, pages 73–79, 2003.

[5] Laurie Williams, E. Michael Maximilien, and Mladen Vouk. Test-driven development as a defect-reduction practice. In *In Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering*, pages 34–45. IEEE Computer Society, 2003.

[6] Artem Marchenko, Pekka Abrahamsson, and Tuomas Ihme. Long-Term Effects of Test-Driven Development A Case Study. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 13–22. Springer Berlin Heidelberg, 2009.

[7] Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337 – 342, 2003.

Chapter 6

Paper B: Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review

Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat
In proceedings of the International Conference on Software Testing (ICST), Berlin, Germany, March 2011

Abstract

Test driven development (TDD) is one of the basic practices of agile software development and both academia and practitioners claim that TDD, to a certain extent, improves the quality of the code produced by developers. However, recent results suggest that this practice is not followed to the extent preferred by industry. In order to pinpoint specific obstacles limiting its industrial adoption we have conducted a systematic literature review on empirical studies explicitly focusing on TDD as well as indirectly addressing TDD. Our review has identified seven limiting factors viz., increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code. The results of this study is of special importance to the testing community, since it outlines the direction for further detailed scientific investigations as well as highlights the requirement of guidelines to overcome these limiting factors for successful industrial adoption of TDD.

6.1 Introduction

Test-driven development (TDD) is an essential part of eXtreme Programming (XP), as proposed by Kent Beck [1]. TDD (referred as test-first programming as well) requires the developers to construct automated unit tests in the form of assertions to define code requirements before writing the code itself. In this process, developers evolve the systems through cycles of test, development and refactoring.

In a recent industrial survey [2], we examined the difference between the preferred and the actual level of usage for a number of contemporary test-related practices. Out of 22 examined practices, TDD gained the highest score of "dissatisfaction"(i.e., the accumulated absolute difference between the preferred and the actual level of usage). Moreover, the preferred level of usage of TDD was significantly higher than the actual level. Hence, the nature of this dissatisfaction could be stated as "Respondents would like to use TDD to a significantly higher extent than they actually do".

Building upon these previous results, the aim of the current study was to investigate potential factors that are limiting the industrial adoption of TDD. Here, a factor could translate to a method, technique, effect, experience, tool, or similar, that either exists, or missing, or is newly introduced in a particular organisation. The specific research question we address in this paper is:

> ***RQ:*** *Which factors could potentially limit the industrial adoption of TDD?*

In order to identify such limiting factors, a systematic literature review of empirical studies on TDD was undertaken. Partly based on concerns of an insufficient number of studies due to publication bias [3], the review was not restricted to studies reporting on failure to implement TDD. Instead, we decided to expand the scope of the study and to systematically search for primary empirical studies of TDD, including (1) studies where TDD was the main focus, (2) studies where TDD was one of the investigated practices, and (3) studies where TDD was used in the experimental setting while investigating something else. In case any of the studies reported issue(s) with any specific factors, this was noted. By qualitatively and quantitatively analysing the reported issues on TDD within the selected papers, we have identified a number of limiting factors. The contributions of this paper are three-fold:

- A qualitative analysis on the effects of a set of factors on TDD, based on reported primary studies

- Identification of a set of factors limiting the adoption of TDD in industry
- Discussions on the implications for research and industry of these factors

The remainder of the paper is organised as follows: Section 6.2 provide details of the research method used, Section 6.3 presents results and analysis while Section 6.4 discuss the findings of our investigation. The paper is concluded by Section 6.5 where conclusion and future work are presented.

6.2 Research Method

A systematic literature review is an empirical study where a research question or hypothesis is approached by collecting and aggregating evidence from a number of primary studies through a systematic search and data extraction process. In this process we followed the guidelines for conducting a systematic literature review proposed by Kitchenham [3].

6.2.1 Search Process

The review process started with the development of a review protocol. This protocol described the purpose of the review, defined the research questions as well as preliminary inclusion and exclusion criteria, and provided details on the search string and the databases in which it would be applied.

As for inclusion and exclusion criteria, the search aimed at identifying full-length English language peer-reviewed empirical studies focusing on TDD, including academic and industrial experiments, case studies, surveys and literature reviews. Short papers (in our case, below six pages), tutorials, work-in-progress papers, keynotes, and pure industrial lessons learned were excluded.

Source	Search Date
IEEExplore	2010-02-12
ACM Digital Library	2010-02-15
ScienceDirect	2010-02-11
EI Compendex	2010-02-12
SpringerLink	2010-02-12
ISI Web of Science	2010-02-11
Wiley Inter Science Journal Finder	2010-02-16

Table 6.1: Searched databases

Scientific databases in the software engineering field were selected based on the aim of getting a wide and exhaustive coverage of published studies on TDD. The list of selected databases is provided in Table 6.1. Within each of these databases, a search was performed using the following Boolean search string:

"tdd" OR "test driven development" OR "test-driven development" OR "test driven design" OR "test-driven design" OR "test first" OR "test-first" OR "integration driven development" OR "integration-driven development"

The resulting list of primary studies was collected in the EndNote reference management program in order to facilitate the paper exclusion process.

6.2.2 Paper Exclusion Process

Following the initial search, which yielded a total of 9.462 papers, exclusion was performed in multiple stages:

1. In the first stage, duplicate papers were removed, and papers were excluded based on formal criteria (e.g., exclusion of short papers) and on title (typically off-topic papers). A total of 509 papers passed this stage.

2. In the second stage, papers were excluded based on abstracts. A total of 150 papers passed this stage.

3. In the third and final exclusion stage, papers were excluded based on full text. In this stage, each paper was read by at least two researchers. To assess the quality and suitability of each study with respect to the review objective, we made use of a review form similar to the screening questions in the quality assessment form used by Dybå and Dingsøyr in their review of empirical studies of agile development [4]. Specifically, we investigated (1) if the paper was a research paper, (2) if it was an evaluation of TDD, (3) if the research aims were clearly stated, and (4) if the paper had an adequate description of context/setting. In order to pass the stage, a paper had to fulfill both criteria (1) and (2) as well as either (3) or (4). A total of 48 papers passed this stage.

Paper exclusion disagreements were resolved through in-depth discussions between the authors.

Extracted study details	
General study information	*Publication type, year, author, etc.*
Study setting	*Academic, industrial or semi-industrial*
Domain of study objective	*Web, business system, embedded system, etc.*
Study type	*Case study, experiment, survey or literature review*
Number of subjects	
Length of study	
Level of experience of subjects	*Novice, medium, experienced*
Focus level of TDD in study	*The main focus of the study is TDD* *One focus of the study is TDD* *TDD is not a focus of the study, but it is used to study something else*

Table 6.2: Extracted study details

6.2.3 Data Extraction Process

In the first step of the data extraction, study details regarding, e.g., study setting and domain, were extracted for all included studies (see Table 6.2). In this step, data extraction was relatively straightforward. However, in the cases where the data of interest was omitted or unclearly stated in the primary study (e.g., failure to mention the level of subjects' previous experience of TDD), the data was omitted from extraction.

The extraction of TDD effects was more complicated. Here, a two-step evolutionary approach was used. In the first step, each selected paper was read by one researcher to identify explicitly stated effects of TDD observed in the study. At this stage, there was no discrimination between negative, neutral or positive effects of TDD. This stage of the review also extracted explicit claims on requirements for a successful adoption of TDD. The reason for not only extracting negative effects of TDD was that we believe that the resulting partial view would diminish the possibilities of performing a balanced analysis of limiting factors. Once the first step of the data extraction was finished, 10 studies were omitted from further analysis. These studies were either studies that were also reported in other included papers, or contained no explicitly reported effects of TDD.

In the second step, the resulting matrix of TDD effects and primary studies was reviewed for consistency by all authors. The aim was to make sure that the claimed effects had been interpreted similarly in the extraction process.

6.2.4 Data Synthesis

Based on the 18 TDD effect areas extracted in the previous step, we defined the limiting factors for the industrial adoption of TDD as effect areas conforming to the following rules:

i. The effect area contained at least two studies with observations of negative effects of or on TDD

ii. The effect area contained more studies with observations of negative effects of or on TDD than it contained studies with observations of positive effects of or on TDD

iii. Negative effects in the effect area were observed in at least one study performed in an industrial setting.

6.3 Results and Analysis

This review identified 48 empirical studies concerning TDD. 38 of those included explicit claims on the effects of TDD. This section provides the study details of the larger set of identified studies, as well as an analysis of the limiting factors of TDD, based on the effects of TDD stated in the primary studies.

6.3.1 Empirical Studies of TDD

An overview of the primary studies on TDD included in our review is given in Table 6.3. Out of the 48 included studies, 25 were experiments, 20 were case studies, 2 were surveys, and one was a mix of a case study and an experiment. 50% of the studies were performed in an academic setting, 46% were studies performed in an industrial setting and 4% were mixed academic/industrial studies. Over half of the included studies (58%) included professional software engineers in the group of study subjects. Most included studies (67%) were studies with TDD as the primary focus of investigation.

Besides the study quality screening used for paper exclusion, we made no further attempt of explicitly evaluating the quality of each included primary study. Even though some additional insights might have been gained by such a quality assessment, we believe that this value would have been limited by the heterogeneity of the included studies.

Study	Setting	Type	Subjects	Focus level
Abrahamson et. al (2005) [5]	Industrial[1]	Case Study	Professionals	TDD explicit primary focus
Bhat & Nagappan (2006) [6]	Industrial	Case Study	Professionals	TDD explicit primary focus
Canfora et. al (2006) [7]	Industrial	Experiment	Professionals	TDD explicit primary focus
Canfora et. al (2006) ISESE [8]	Industrial	Experiment	Professionals	TDD explicit primary focus
Cao & Ramesh (2008) [9]	Industrial	Case Study	Professionals	TDD explicit focus, but not main focus
Chien et. al (2008) [10]	Academic	Experiment	Students	TDD explicit focus, but not main focus
Damm & Lundberg (2006) [11]	Industrial	Case Study	Professionals	TDD explicit primary focus
Damm & Lundberg (2007) [12]	Industrial	Case Study	Professionals	TDD explicit primary focus
Domino et. al (2007) [13]	Academic	Experiment	Students	TDD explicit focus, but not main focus
Domino et. al (2003) [14]	Academic	Experiment	Students	TDD not in focus, but used in study setup
Erdogmus et. al (2005) [15]	Academic	Experiment	Students	TDD explicit primary focus
Filho (2006) [16]	Academic	Experiment	Students	TDD not in focus, but used in study setup
Flohr & Schneider (2005) [17]	Academic	Experiment	Students	TDD explicit primary focus
Flohr & Schneider (2006) [18]	Academic	Experiment	Students	TDD explicit primary focus
George & Williams (2003) [19]	Industrial	Experiment	Professionals	TDD explicit primary focus
Geras et. al (2004) [20]	Academic	Experiment	Professionals	TDD explicit primary focus
Gupta & Jalote (2007) [21]	Academic	Experiment	Students	TDD explicit primary focus
Hfer & Philipp (2009) [22]	Academic	Experiment	Mixed	TDD explicit primary focus
Huang & Holcombe (2009) [23]	Academic	Experiment	Students	TDD explicit primary focus
Janzen & Saiedian (2006) [24]	Academic	Experiment	Students	TDD explicit primary focus
Janzen & Saiedian (2008) [25]	Mixed	Exp./Case Study	Mixed	TDD explicit primary focus
Janzen et. al (2007) [26]	Academic	Experiment	Professionals	TDD explicit primary focus
Kobayashi et. al (2006) [27]	Industrial	Case Study	Professionals	TDD explicit focus, but not main focus
Kollanus & Isomttnen (2008) [28]	Academic	Experiment	Students	TDD explicit primary focus
Layman et. al (2006) [29]	Industrial	Case Study	Professionals	TDD explicit focus, but not main focus
LeJeune (2006) [30]	Academic	Case Study	Students	TDD explicit focus, but not main focus

Study	Setting	Type	Subjects	Focus level
Huang et. al (2007) [31]	Academic	Experiment	Students	TDD explicit focus, but not main focus
Madeyski (2006) [32]	Academic	Experiment	Students	TDD explicit focus, but not main focus
Madeyski (2007) [33]	Academic	Experiment	Students	TDD not in focus, but used in study setup
Madeyski (2008) [34]	Academic	Experiment	Students	TDD not in focus, but used in study setup
Madeyski (2010) [35]	Academic	Experiment	Students	TDD explicit primary focus
Madeyski & Szaa (2007) [36]	Industrial	Case Study	Professionals	TDD explicit primary focus
Marchenko et. al (2009) [37]	Industrial	Case Study	Professionals	TDD explicit primary focus
Maximilien & Williams (2003) [38]	Industrial	Case Study	Professionals	TDD explicit primary focus
Mišić (2006) [39]	Mixed	Survey	Mixed	TDD explicit focus, but not main focus
Müller & Hagner (2002) [40]	Academic	Experiment	Students	TDD explicit primary focus
Müller & Hfer (2007) [41]	Academic	Experiment	Mixed	TDD explicit primary focus
Nagappan et. al (2008) [42]	Industrial	Case Study	Professionals	TDD explicit primary focus
Salo & Abrahamsson (2007) [43]	Industrial[1]	Case Study	Professionals	TDD not in focus, but used in study setup
Sanchez et. al (2007) [45]	Industrial	Case Study	Professionals	TDD explicit primary focus
Sfetsos et. al (2006) [46]	Industrial	Survey	Mixed	TDD explicit focus, but not main focus
Sherrell & Robertson (2006) [47]	Academic	Case Study	Students	TDD explicit focus, but not main focus
Siniaalto & Abrahamsson (2007) [48]	Industrial	Case Study	Professionals	TDD explicit primary focus
Siniaalto & Abrahamsson (2008) [49]	Industrial	Case Study	Professionals	TDD explicit primary focus
Slyngstad et. al (2008) [50]	Industrial	Case Study	Professionals	TDD explicit primary focus
Wastnus & Gross (2007) [51]	Industrial	Case Study	Professionals	TDD explicit primary focus
Williams et. al (2003) [52]	Industrial	Case Study	Professionals	TDD explicit primary focus
Vu et. al (2009) [53]	Academic	Experiment	Students	TDD explicit primary focus

Table 6.3: Empirical Studies of TDD

[1] Close-to-Industry setting as defined in [44]

6.3.2 Reported Effects of and on TDD

In order to identify limiting factors of industrial adoption of TDD, all included studies were searched for explicit claims on effects of TDD (i.e., cause-effect relationships where TDD was the causing factor), as well as explicit claims on effects on TDD (i.e., cause-effect relationships where some factor caused an effect on the way TDD was performed). We denote these effect areas of TDD, and Table 6.4 presents the 18 effect areas found in the search. A total count for the number of studies making claims regarding each effect is also given in the table.

Sl.No.	Description	Count
1	Development time	18
2	Experience/knowledge	4
3	Design	3
4	Refactoring	2
5	Skill in testing	3
6	TDD adherence	8
7	Code quality	18
8	Cost	1
9	Code coverage	8
10	Complexity	7
11	Time for feedback	5
12	Domain and tool specific issues	10
13	Code size	3
14	Perceptions	15
15	Communication & (customer) collaboration	1
16	Legacy code	2
17	Defect reproduction	1
18	Documentation	1

Table 6.4: Areas of Effect of TDD

Again, note that effects were included in this data extraction regardless of whether they were mentioned in a positive, neutral or negative context. As mentioned above, the collection of included primary studies exhibited great heterogeneity. As a consequence, description of TDD effects ranged from purely quantitative data (e.g., ratio scale metrics on code complexity [25][51] or development time [6][19]), to qualitative data based on subjects' responses to open-ended survey questions (e.g., nominal or ordinal scale claims on per-

ception of TDD [30][43]). Consequently, the items in the resulting list of effect areas are not necessarily unique and independent. As an example, effect areas like design, refactoring and complexity are highly related. When doubtful, we have chosen not to group effect areas, as this would result in a less rich information from which to derive the limiting factors of TDD.

A more detailed view of the areas of effect of TDD with respect to the primary studies is provided in Table 6.6.

6.3.3 Factors Limiting Industrial Adoption of TDD

Based on the effect areas presented in the previous section, we identify seven limiting factors (LF1-LF7) for industrial adoption of TDD. An overview of these factors is given in Table 6.5. We now describe each of these limiting factors in detail together with the observations from the primary studies as well as providing motivations for their inclusions.

Label	Description
LF1	Development time
LF2	Experience/knowledge
LF3	Design
LF4	Skill in testing
LF5	TDD adherence
LF6	Domain and tool specific issues
LF7	Legacy code

Table 6.5: Limiting Factors for TDD Adoption

Study \ Effect	Development time	Experience/knowledge	Design	Refactoring	Skill in testing	TDD adherence	Code quality	Cost	Code coverage	Complexity	Time for feedback	Domain and tool specific issues	Code size	Perceptions	Communication & customer collaboration	Legacy code	Defect reproduction	Documentation
Abrahamson et. al (2005) [5]		-				-						-						
Bhat & Nagappan (2006) [6]	-						+											
Cao & Ramesh (2008) [9]	-					-					+							
Damm & Lundberg (2007) [12]												-						
Domino et. al (2007) [13]						!	=											
Erdogmus et. al (2005) [15]							=											
Filho (2006) [16]							+											
Flohr & Schneider (2005) [17]												-						
Flohr & Schneider (2006) [18]	+								=					!				
George & Williams (2003) [19]	-						+		+					+				
Geras et. al (2004) [20]	!				!		!											
Gupta & Jalote (2007) [21]	+		-															
Höfer & Philipp (2009) [22]		!		!		!												
Huang & Holcombe (2009) [23]	-						-											
Janzen & Saiedian (2006) [24]	+												-	+				
Janzen & Saiedian (2008) [25]										+			+	!				
Janzen et. al (2007) [26]									+					+				
Kobayashi et. al (2006) [27]	+						+	+										
Kollanus & Isomttnen (2008) [28]			-		-							-		!				
Layman et. al (2006) [29]						-						-				-		
LeJeune (2006) [30]	-						+							+				
Madeyski (2010) [35]									=					-				

Study \ Effect	Development time	Experience/knowledge	Design	Refactoring	Skill in testing	TDD adherence	Code quality	Cost	Code coverage	Complexity	Time for feedback	Domain and tool specific issues	Code size	Perceptions	Communication & customer collaboration	Legacy code	Defect reproduction	Documentation
Marchenko et. al (2009) [37]	-		-				+			+		!		-!		-	-	+
Maximilien & Williams (2003) [38]	-/=	!					+				+			+				
Mišić (2006) [39]						+												
Müller & Hagner (2002) [40]	-/=																	
Müller & Höfer (2007) [41]						!			!		!							
Nagappan et. al (2008) [42]	-					!	+											
Salo & Abrahamsson (2007) [43]		-										-		+				
Sanchez et. al (2007) [45]	-						+			+								
Sfetsos et. al (2006) [46]					-						+	-			!			
Sherrell & Robertson (2006) [47]														-				
Siniaalto & Abrahamsson (2007) [48]									+	-								
Siniaalto & Abrahamsson (2008) [49]							!			!								
Slyngstad et. al (2008) [50]				+			+											
Wastnus & Gross (2007) [51]	+						+		+	+	+	-		+/-				
Williams et. al (2003) [52]	=						+							+				
Vu et. al (2009) [53]	-						+		-	+/!		-	!	+				

Table 6.6: Mapping Between Effect Observations and Primary Studies

Table legend:

+ positive mentioning of a particular effect
- negative mentioning of a particular effect
= no effect was reported for a particular effect
! effect was mentioned as an important observation

LF1: Increased development time

Description: By development time, we refer to the time required to implement a given set of requirements. Time required for development of software product is relatively easy to measure. It is however a matter of discussion whether the time for corrective re-work (e.g., based on failure reports from later testing stages) should be included in the development time or not.

Observations: Nine included primary studies in the review reported negative experience with respect to the time for development. Six were industrial studies with professionals (five case studies and one experiment) and three were academic studies with students (two experiments and one case study). Five studies did report positive effect on development time when using TDD, but this was mostly when the overall project time was captured.

Discussion: Development time could be considered a business-critical factor for adopting new practices within an organisation. Depending on the maturity of the organization, an up-front loss (in this case, increased development time) might overshadow a long-term gain (e.g., decreased overall project time, or increased product quality both of which were reported in many of our included studies). Hence, internal organizational pressure might risk the proper usage of TDD.

LF2: Insufficient TDD experience/knowledge

Description: By TDD experience/knowledge, we refer to the degree of practical experience, as a developer or similar. or theoretical insight in TDD.

Observations: Two industrial case studies with professional developers attributed problems of implementing TDD to lack of TDD education/experience. Moreover, two other studies report significant differences in the way of applying TDD between experienced TDD developers and novice TDD developers.

Discussion: When observing collected data from the included primary studies, we noticed that participants in the experiments (either students or professionals) were mostly provided with some training or tutorial on how to perform TDD. In several cases [43], the knowledge improved as participants would progress with the experiment. We expect that lack of knowledge or experience with TDD could create problems in its adoption.

LF3: Insufficient design

Description: Design, in this context, refers to the activity of structuring (or re-structuring) the system or software under development or in evolution in order to avoid architectural problems, and to improve architectural quality. Detailed up-front software design is a common practice of plan-driven development methodologies. TDD emphases on a small amount of up-front design, and frequent refactoring to keep the architecture from erosion.

Observations: Three primary studies reported architectural problems when using TDD. These were two academic experiments with students and one industrial case study with professionals.

Discussion: There is no massive empirical support that the lack of design should be considered as a limiting factor for industrial adoption of TDD. However, there are a handful of studies reporting problems regarding lack of design in TDD, particularly in the development of larger, more complex systems. Moreover, the lack of upfront design has been one of the main criticisms of TDD since its introduction and even if the evidence supporting this criticism is sparse, so is the evidence contradicting it [57].

LF4: Insufficient developer testing skills

Description: By developer testing skill, we refer to the developer's ability to write efficient and effective automated test cases.

Observations: Two of the included primary studies report negative experiences with developers' testing. One study is an academic experiment with student subjects, where it is reported that students expressed difficulties to come up with good test cases. The other study is an industrial case study with mixed professional and student subjects where lack of developer testing skills was stated as a limiting factor.

Discussion: Since TDD is a design technique where the developer undertakes development by first creating test cases and then writes code that makes the test cases pass, it relies on the ability of the developer to produce sufficiently good test cases. Additionally, Geras [20] reports on the risk it brings to adopt TDD without having adequate testing skills and knowledge. We find it interesting that there are no explicit investigations of the quality of test cases produced by developers in TDD.

LF5: Insufficient adherence to the TDD protocol

Description: By adherence to the TDD protocol, we refer to the degree to which the steps of the TDD practice are followed. For example, are test cases always created and exercised to failure before the corresponding code is written? TDD is a defined practice with fairly exact guidelines on how it should be executed.

Observations: Related to this limiting factor, there are two types of observations that are of relevance. First, three industrial case studies, with professionals as subjects, report negative experiences with developer adherence to the TDD protocol. Reasons for abandoning the TDD protocol included time pressure, lack of discipline, and shortage of perceived benefits. Second, two additional industrial case studies, with professionals as subjects, reported correlations between low TDD adherence and low quality. It is noteworthy that these observations were made in organizations where TDD was the preferred development method.

Discussion: The combined view of the five above mentioned industrial case studies motivate the inclusion of the lack of TDD adherence as a limiting factor. Basically, the studies state that (1) it is important to adhere to the TDD protocol, and (2) developers do stray from the protocol in several situations. It is however far from certain that there is a clean-cut cause-effect relationship between low TDD adherence and low quality. Not unlikely, confounding factors (e.g., tight development deadlines) might lead to both low TDD adherence and poor quality.

LF6: Domain- and tool-specific limitations

Description: By domain- and tool-specific limitations, we refer to technical problems in implementing TDD (e.g., difficulties in performing automated testing of GUIs). Generally, the TDD practice requires some tool support in the form of automation framework for test execution.

Observations: Nine studies reported negative experiences with respect to domain and tool-specific issues. Five of them were industrial case studies with professionals as subjects, one was an industrial survey with both student and professional respondents and three were academic experiments with student subjects. The single most reported issue is the problem of automatically testing GUI applications, but also networked applications seem to be problematic in terms of automated testing.

Discussion: Proper tool support for test automation is vital for the successful adoption of TDD. With the wide variety of studies reporting domain- and tool-specific issues as a limiting factor in the adoption of

TDD, the factor would be difficult to ignore.

LF7: Legacy code

Description: By legacy code, we refer to the existing codebase in a development organization. Legacy code often represent decades of development efforts and investments, and serve as a backbone both in existing and future products.

Observation: Two industrial case studies with professionals as subjects report problems with handling the legacy codebase in an adoption of TDD. Particularly, the automated regression test suites on unit level (which are natural consequences of long-term TDD development), are often missing for legacy code.

Discussion: TDD, in its original form, does not discuss how to handle legacy code. Instead, the method seems to assume that all code is developed from scratch, using TDD as the development method. As this is seldom the case in large development organization, adoption of TDD might be problematic. A lack of automated regression suites for legacy code hampers the flexibility provided by the quick feedback on changes provided by the regression suites, and may leave developers more anxious about how new changes may unexpectedly affect existing code.

6.4 Discussion

In this section we are discussing threats to validity of our research as well as implications of our results on research and industry.

6.4.1 Threats to Validity

Typically, four types of validity are discussed in empirical research (i.e., *construct validity, internal validity external validity and reliability*) [54]. Below, the threats to these validities in our study are discussed.

Construct Validity refers to the correctness in the mapping between the theoretical constructs that are to be investigated, and the actual observations of the study. In a systematic review, the validity of the study constructs is inherited from the construct validity in the included primary studies. In our case, this validity threat concerned both the actual treatment of the primary studies (i.e., TDD) and the effect on study outcomes (e.g, quality or development time).

First, in order to measure the effects of TDD in an empirical study, one must be sure that TDD is actually used within the study. This problem was handled

differently in different studies. Some studies merely assumed that TDD was used by the set of subjects as instructed [16], some studied used manual supervision [5], and some studies used elaborate tools to ensure TDD adherence [20]. Second, with respect to outcome measures (i.e., the effects of TDD), the construct validity is different for different constructs. Most metrics for, e.g., complexity are formally defined and measured, and are hence not subjects to threats to construct validity. However, constructs like design quality and developer skill are subject to interpretation in each primary study. In the review, we sought to mitigate this threat by performing the data extraction in two phases, with the second phase focusing on a conformance in the interpretation of primary study constructs between the authors.

Internal Validity concerns the proper analysis of data. Given the heterogeneity of the included primary studies in the review, internal validity is a subject of concern, particularly in statistical analysis of the extracted data. However, we draw no generalized statistical conclusions regarding the effects of TDD. Rather, our contribution is a set of directions for future research and industrial guidelines, based on a qualitative analysis of the extracted data.

External Validity relates to the possibility to generalize the study results outside its scope of investigation. The variety of study setting, type and domain serves to limit the external validity threat of the review, particularly in the cases where limiting factors are found across several studies in different domains. Also, by collecting study details, we had the possibility to differentiate results based on particular study details.

Reliability concerns the degree of certainty with which a replication of this study, e.g., by a different set of researchers, would yield the same study outcome. As the search strategy, as well as the inclusion and exclusion criteria, is explicitly given in this study, the main reliability threat concerns the analysis resulting in the aggregation from reported effects of TDD to limiting factors. Particularly effects with low construct validity may be interpreted differently in replicated reviews. Hence, we have sought to describe the research process, including the data analysis, in a transparent manner.

6.4.2 Implications for Research

Test driven development was and still is under constant investigation of researchers who are providing evidence of claimed benefits which this practice can bring to a software development team. These benefits can be also seen in our mapping table (Table V) between effect observations and primary studies. Most noticeable positive effect is a code quality improvement which is

one of the reasons why TDD is gaining interest. Also we can see that primary studies are reporting a positive perception of participants towards TDD. This is something our previous study [2] also revealed.

Having those two benefits empirically addressed (quality improvement and positive perception of practice) we propose that the next empirical evaluation on TDD should include a direct investigation on the impact of the limiting factors we presented in Section 6.3.

Next studies could focus on a limiting factor of development time (LF1) together with the lack of TDD experience factor (LF2) to investigate if the actual learning curve could be generating additional time for the development. However, it is important for researchers to clearly state if increased development time was reported during unit development or it is reflected on overall project. This is something we had difficulties extracting from primary studies.

By providing more complex algorithms or working in a different domain of investigation (safety-critical systems on embedded device for example) researchers could investigate how lack of up-front design (LF3) influence adoption of TDD. Even in a student experiment setup, Kollanus [28] noted that slightly complex application required more of up-front design.

TDD is a development technique which requires from developers to write test cases. We noticed that primary studies are not directly investigating how these test cases are designed and whether designing test cases for TDD is different from how experienced testers are performing it. By investigating LF4 with independent teams of experienced testers and developers with the focus on efficiency and quality of test design, researchers could gain insights on issues such as 1) whether lack of quality in tests is limiting adoption of TDD, and 2) the right level of testing education required for developers to perform TDD.

Regarding LF5, TDD adherence needs to be further evaluated. A first step would be to make sure that all studies examining the effects of TDD also has some means of measuring TDD adherence in the experimental setting. Particularly in industrial case studies, it would be valuable to investigate TDD adherence over time, with variations in, e.g., TDD experience, workload and type of development task. Such observations should be correlated with resulting measurements on quality and development time.

Regarding the more technical limiting factors (i.e., the domain- and tool-specific issues (LF6), and the lack of automated regression suites for legacy code (LF7)), it is our belief that research could contribute with improved methods and techniques for different aspects of test automation, including automated test case generation, test case execution and test case evaluation.

6.4.3 Implications for Industry

This review identifies a set of potentially limiting factors of industrial adoption of TDD, based on aggregated observations from TDD usage in various different settings. Consequently, a set of industrial guidelines can be derived from the study results.

First, the limiting factors that are controllable and specific to the adopting organization should be taken into account prior to TDD adoption. Specifically, proper training on TDD practice (relating to LF2) and test case design (relating to LF4) should be provided before the adoption. Additionally, strategic recruitments of experienced TDD developers, who might serve as TDD mentors could limit the problems related to lack of TDD experience. Moreover, TDD adoption should be considered in the light of the organizational domain (relating to LF6). For example, are the developed systems heavy on graphical user interaction? Is there proper tool support in the existing development infrastructure for the level of test automation required by TDD? In addition, if the adopting organization includes a significant legacy codebase lacking automated regression test suites (relating to LF7), there should be a strategy of how to handle this that does not collide with the TDD development protocol.

Second, the limiting factors that are general and TDD-inherent should be considered and monitored in the adopting organization (with respect to that organization's motives for adopting TDD). Specifically, it should be considered whether a small increase in (unit-level) development time, if observed, would be acceptable to reach other expected benefits of TDD (relating to LF1). Moreover, it would be advisable to track the architectural quality (both based on metrics of architectural quality attributes and developer perception) to ensure that the lack of upfront design does not lead to architectural erosion (relating to LF3). In addition, although the enforcing TDD protocol upon the developers might not be a good idea, it might still be advisable to keep track of the effects on, e.g., quality, in situations where the TDD protocol is not followed (relating to LF5).

6.5 Conclusion

In this paper we present our analysis results from a systematic review of the empirical studies reported in the literature on the effects of various factors on Test Driven Development. We identified 18 effects, out of which 7 were deemed as limiting factors on the industrial adoption of TDD. These factors were increased development time, insufficient TDD experience/knowledge, lack of up-

front design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code. We also provided reasons for their inclusion as well as discussions on the various implications of these factors. We also outlined the future research and industrial challenges in the light of these findings.

From the perspective of the software testing community, this study throws open the following interesting research challenges to be addressed:

- What is the optimum level of testing knowledge essential for a TDD developer to be efficient?
- Are there any fundamental changes warranted by the TDD approach in the test design techniques?
- How to integrate the TDD perspective of testing and the testers' perspective of traditional development to provide a unified theory for bringing synergy between development and testing efforts in a more productive manner?
- In what new roles [55] and how testers could contribute with their knowledge and experience in the adoption of TDD process within an organisation?

6.6 Acknowledgments

This work was supported by MRTC (Mlardalen Real-Time Research Centre) and the SWELL (Swedish software Verification & Validation ExceLLence) research school. Authors would like to express their gratitude to Dr. Stig Larsson for valuable comments to the review protocol.

Bibliography

[1] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[2] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. An Industrial Survey on Contemporary Aspects of Software Testing. In *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST)*, pages 393–401, 2010.

[3] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[4] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833 – 859, 2008.

[5] Pekka Abrahamsson, Antti Hanhineva, and Juho Jäälinoja. Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development. In *Business Agility and Information Technology Diffusion*, volume 180/2005 of *IFIP International Federation for Information Processing*, pages 227–243. Springer Boston, 2006.

[6] Thirumalesh Bhat and Nachiappan Nagappan. Evaluating the efficacy of test-driven development: Industrial case studies. In *ISCE'06 - 5th ACM-IEEE International Symposium on Empirical Software Engineering, September 21, 2006 - September 22, 2006*, volume 2006 of *ISCE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empir-*

ical Software Engineering, pages 356–363, Rio de Janeiro, Brazil, 2006. Association for Computing Machinery.

[7] Gerardo Canfora, Aniello Cimitile, Felix Garcia, Mario Piattini, and Corrado Aaron Visaggio. Evaluating advantages of test driven development: A controlled experiment with professionals. ISCE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering, 2006.

[8] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio. Productivity of test driven development: A controlled experiment with professionals. *Product-Focused Software Process Improvement, Proceedings*, 4034:383–388, 2006.

[9] Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(Compendex):60–67, 2008.

[10] L. R. Chien, D. J. Buehrer, C. Y. Yang, and C. M. Chen. An Evaluation of TDD Training Methods in a Programming Curriculum. *2008 Ieee International Symposium on It in Medicine and Education, Vols 1 and 2, Proceedings*, pages 660–665, 2008.

[11] Lars-Ola Damm and Lars Lundberg. Results from introducing component-level test automation and Test-Driven Development. *Journal of Systems and Software*, 79(7):1001–1014, 2006.

[12] Lars-Ola Damm and Lars Lundberg. Quality impact of introducing component-level test automation and test-driven development. *Software Process Improvement, Proceedings*, 4764:187–199, 2007.

[13] Madeline Domino, Rosann Collins, and Alan Hevner. Controlled experimentation on adaptations of pair programming. *Information Technology and Management*, 8:297–312, 2007.

[14] Madeline Ann Domino, Rosann Webb Collins, Alan R. Hevner, and Cynthia F. Cohen. Conflict in collaborative software development, 2003.

[15] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31:226–237, 2005.

[16] Wilson P. Paula Filho. Quality gates in use-case driven development. In *Proceedings of the 2006 international workshop on Software quality*, WoSQ '06, pages 33–38, New York, NY, USA, 2006. ACM.

[17] Thomas Flohr and Thorsten Schneider. An XP Experiment with Students Setup and Problems. In Frank Bomarius and Seija Komi-Sirviö, editors, *Product Focused Software Process Improvement*, volume 3547 of *Lecture Notes in Computer Science*, pages 95–111. Springer Berlin / Heidelberg, 2005.

[18] Thomas Flohr and Thorsten Schneider. Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings. In Jrgen Münch and Matias Vierimaa, editors, *Product-Focused Software Process Improvement*, volume 4034 of *Lecture Notes in Computer Science*, pages 305–318. Springer Berlin / Heidelberg, 2006.

[19] B. George and L. Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337–342, 2003.

[20] A. Geras, M. Smith, and J. Miller. A Prototype Empirical Evaluation of Test Driven Development. In *Proceedings of the Software Metrics, 10th International Symposium*, pages 405–416, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Atul Gupta and Pankaj Jalote. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 285–294, Washington, DC, USA, 2007. IEEE Computer Society.

[22] Andreas Höfer and Marc Philipp. An Empirical Study on the TDD Conformance of Novice and Expert Pair Programmers. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 33–42. Springer Berlin Heidelberg, 2009.

[23] Liang Huang and Mike Holcombe. Empirical investigation towards the effectiveness of Test First programming. *Inf. Softw. Technol.*, 51:182–194, January 2009.

[24] David S. Janzen and Hossein Saiedian. On the Influence of Test-Driven Development on Software Design. *Software Engineering Education and Training, Conference on*, pages 141–148, 2006.

[25] D. S. Janzen and H. Saiedian. Does test-driven development really improve software design quality? *IEEE Software*, 25(2):77–84, 2008.

[26] David S. Janzen, Clark S. Turner, and Hossein Saiedian. Empirical software engineering in industry short courses. Software Engineering Education Conference, Proceedings, pages 89–96, 2007.

[27] Osamu Kobayashi, Mitsuyoshi Kawabata, Makoto Sakai, and Eddy Parkinson. Analysis of the interaction between practices for introducing XP effectively. ICSE '06, pages 544–550, 2006.

[28] Sami Kollanus and Ville Isomöttönen. Understanding TDD in academic environment: experiences from two experiments. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 25–31, New York, NY, USA, 2008. ACM.

[29] Lucas Layman, Laurie Williams, and Lynn Cunningham. Motivations and measurements in an agile case study. *Journal of Systems Architecture*, 52(11):654–667, 2006.

[30] Noel F LeJeune. Teaching software engineering practices with Extreme Programming. *J. Comput. Small Coll.*, 21(3):107–117, 2006.

[31] Liang Huang, C. Thomson, and M. Holcombe. How good are your testers? An assessment of testing ability. In *TAIC-PART 2007*, pages 82 –88, 2007.

[32] Lech Madeyski. [the impact of pair programming and test-driven development on package dependencies in object-oriented design - an experiment.

[33] Lech Madeyski. On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests. *Product-Focused Software Process Improvement, Proceedings*, 4589:207–221, 2007.

[34] Lech Madeyski. Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites. *Software Process: Improvement and Practice*, 13(3):281–295, 2008.

[35] Lech Madeyski. The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment. *Inf. Softw. Technol.*, 52:169–184, February 2010.

[36] Lech Madeyski and Ł ukasz Szał a. The impact of test-driven development on software development productivity - An empirical study. *Software Process Improvement, Proceedings*, 4764:200–211, 2007.

[37] Artem Marchenko, Pekka Abrahamsson, and Tuomas Ihme. Long-Term Effects of Test-Driven Development A Case Study. In Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 13–22. Springer Berlin Heidelberg, 2009.

[38] E. M. Maximilien and L. Williams. Assessing test-driven development at IBM. *25th International Conference on Software Engineering, Proceedings*, pages 564–569, 2003.

[39] Vojislav B. Mišić. Perceptions of extreme programming: an exploratory study. *SIGSOFT Softw. Eng. Notes*, 31:1–8, March 2006.

[40] M.M. Müller and O. Hagner. Experiment about test-first programming. *Software, IEE Proceedings* -, 149(5):131 – 136, October 2002.

[41] Matthias Müller and Andreas Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12:593–615, 2007.

[42] Nachiappan Nagappan, E. Maximilien, Thirumalesh Bhat, and Laurie Williams. Realizing quality improvement through test driven development: results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3):289–302, 2008.

[43] Outi Salo and Pekka Abrahamsson. An iterative improvement process for agile software development. *Software Process: Improvement and Practice*, 12(1):81–100, 2007.

[44] Outi Salo and Pekka Abrahamsson. Empirical Evaluation of Agile Software Development: A Controlled Case Study Approach. In *5th International Conference on Product Focused Software Process Improvement*, 2004.

[45] Julio Cesar Sanchez, Laurie Williams, and E. Michael Maximilien. On the sustained use of a test-driven development practice at IBM. Proceedings - AGILE 2007, pages 5–14, 2007.

[46] P. Sfetsos, L. Angelis, and I. Stamelos. Investigating the extreme programming system - An empirical study. *Empirical Software Engineering*, 11(2):269–301, 2006.

[47] Linda B. Sherrell and Jeff J. Robertson. Pair programming and agile software development: experiences in a college setting. *J. Comput. Small Coll.*, 22(2):145–153, 2006.

[48] Maria Siniaalto and Pekka Abrahamsson. A comparative case study on the impact of test-driven development on program design and test coverage. Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007, pages 275–284, 2007.

[49] M. Siniaalto and P. Abrahamsson. Does test-driven development improve the program code? Alarming results from a comparative case study. *Balancing Agility and Formalism in Software Engineering*, 5082:143–156, 2008.

[50] Odd Petter N. Slyngstad, Jingyue Li, Reidar Conradi, Harald Rnneberg, Einar Landre, and Harald Wesenberg. The impact of test driven development on the evolution of a reusable framework of components - An industrial case study. Proceedings - The 3rd International Conference on Software Engineering Advances, ICSEA 2008, pages 214–223, 2008.

[51] H. Wastnus and H. G. Gross. Evaluation of test-driven development - An industrial case study. *Enase 2007: Proceedings of the Second International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 103–110, 2007.

[52] Laurie Williams, E. Michael Maximilien, and Mladen Vouk. Test-driven development as a defect-reduction practice. In *In Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering*, pages 34–45. IEEE Computer Society, 2003.

[53] John Huan Vu, Niklas Frojd, Clay Shenkel-Therolf, and David S. Janzen. Evaluating Test-Driven Development in an Industry-Sponsored Capstone Project. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 229–234, Washington, DC, USA, 2009. IEEE Computer Society.

[54] Claes Wohlin, Per Runesson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, 2000.

[55] Adnan Causevic, Abdulkadir Sajeev, and Sasikumar Punnekkat. Redefining the role of testers in organisational transition to agile methodologies. In *International Conference on Software, Services & Semantic Technologies (S3T)*, October 2009.

Chapter 7

Paper C: Impact of Test Design Technique Knowledge on Test Driven Development: A Controlled Experiment

Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat
In submission

Abstract

Agile development approaches, in spite of the skepticism on their appropriateness in high reliability applications, are increasingly being followed and favored by the industry. Test Driven Development (TDD) is a key agile practice and recent research results suggest that the successful adoption of TDD depends on different limiting factors, one of them being insufficient developer testing skills. The goal of this paper is to investigate if developers who are educated on general testing knowledge will be able to utilize TDD more effectively. We conducted a controlled experiment with master students during the course on Software Verification & Validation (V&V) where source code and test cases created by each participant during the labs as well as their answers on a survey questionnaire were collected and analyzed.

Descriptive statistics indicate improvements in statement coverage. However, no statistically significant differences could be established between the pre- and post-course groups of students. By qualitative analysis of students' tests, we noticed a lack of negative test cases resulting in non-detection of bugs. Students did show preference towards TDD in surveys. The experiment was conducted in an academic setting with student subjects, potentially threatening the external validity of the results.

Although further research is required to fully establish this, we believe that identifying specific testing knowledge which is complementary to the testing skills of a new TDD developer would enable developers to perform their tasks in a more efficient manner resulting in improved reliability of software products.

7.1 Motivation

Test Driven Development (TDD), also known as test-first programming, is an essential part of eXtreme Programming (XP) [1]. TDD requires the developers to construct automated unit tests in the form of assertions to define code requirements before writing the code itself. In this process, developers evolve the systems through cycles of test, development and refactoring. In a recent industrial survey [2], we examined the difference between the preferred and the actual level of usage for several test-related practices. Among the 22 examined practices, surprisingly, TDD gained the highest score of 'dissatisfaction'. This means that the accumulated absolute difference between the preferred and the actual levels of usage was highest in the case of TDD. The nature of this dissatisfaction could be stated as "Respondents would like to use TDD to a significantly higher extent than they actually do currently".

Subsequently we explored the current body of knowledge through an empirical systematic literature review [3] to identify the limiting factors which prevents the successful adoption of TDD. Insufficient developer testing skill was identified as one of the important limiting factors as part of the study.

7.1.1 Problem Statement

TDD in its essence teaches developers how to perform software development providing some indirect basic testing skills, for example based on *positive testing* (i.e. testing to show that the software "works" using valid input). We are interested in identifying specific testing knowledge which is complementary to the testing skills of a new TDD developer. We believe that such a strategy would enable developers to perform their tasks in a more efficient manner resulting in higher quality of software products.

7.1.2 Research Objective

Using the form originated from GQM [4], the research objective of this study can be expressed as follows:

> To analyze *the effect of testing knowledge on TDD* for the purpose of *evaluation of factors affecting the outcome of TDD* with respect to the *factors' limiting effect on the usage of TDD* from the point of view of *the software developer* in the context of *eXtreme Programming software development.*

7.1.3 Context

To perform analysis with respect to the above objective, an experiment was organised as lab activities with master students enrolled in the Software Verification and Validation course at Mälardalen University during the autumn semester of 2010.

7.1.4 Paper Outline

This paper is structured according to the reporting guidelines provided by Jedlitschka and Pfahl [5] (although some minor deviations from the reporting guidelines were made). In section 7.2 we present the related research works followed by the experimental design in section 7.3. Section 7.4 presents the details of execution of our experiment. The treatment and analysis of the collected data are given in section 7.5. In section 7.6, we present statistical inferences followed by conclusions and future research planned in section 7.7.

7.2 Related Work

Test-driven development is a practice derived from experience which makes it very difficult to prove its efficiency in a formal way. This is one of the reasons why many experiments on TDD are conducted in order to provide empirical evidence of its claimed quality improvements.

In this section we present related work on empirical investigations of TDD identified in our recent systematic literature review [3], grouped w.r.t two aspects: (i) related to testing knowledge and (ii) general experiments on TDD.

7.2.1 TDD and testing knowledge

Sfetsos et al. [6] performed an industrial survey on advantages and difficulties that software companies experienced when applying XP. Test-first was among the investigated practices. During interviews, developers pointed to difficulties in writing tests at the very beginning of the project.

Geras et al. [7] performed an experiment with professionals in academic environment providing subjects with two programs for development, one using test-last and one using test-first process. One of the conclusions made from the experiment is that without adequate training and having proper testing skills it is risky to adopt TDD.

Kollanus & Isomöttönen [8] analysed students' perceptions and difficulties on TDD in an educational context experiment. As part of their conclusions they present different difficulties students had when designing tests. Generally, students find it difficult to design appropriate test cases and to design tests in small steps.

7.2.2 Experiments in TDD

In Table 7.1 we present experiments in TDD selected from [3] outlining experiment environment (industrial or academic) and type of subjects (students, professionals or mixed). A brief description of the aim and major results of each TDD study is also presented.

7.3 Experimental Design

This section details the design of the experiment. Further practical experiment setup information, e.g., for replication purposes, can be found at the first author's webpage[1].

7.3.1 Goals, Hypotheses, Parameters, and Variables

The goal of the experiment was to test the effect of knowledge in software testing on *development speed*, *artefact quality* and *developer perception* when using TDD. In order to do so, the following null and alternative hypotheses were formulated:

- **Development Speed:**
  - $\mathbf{H}^s{}_0$. When using TDD, there is no significant difference between the development speed of developers with or without knowledge in software testing.
  - $\mathbf{H}^s{}_a$. When using TDD, developers with knowledge in software testing develop faster.

[1] http://www.mrtc.mdh.se/~acc01/tddexperiment/

Authors	Year	Experiment settings	Subjects
Müller & Hagner [9]	2002	Academic	Students
Aim: To evaluate benefits of test-first programming compared to traditional approach.			
Results: Test-first does not accelerate programming, produced programs are not more reliable but test-first support better understanding of program.			
George & Williams [10]	2003	Industrial	Professionals
Aim: To evaluate quality improvements of test-driven development compared to a waterfall-like approach.			
Results: Test-driven development produces higher quality code with the tendency of developers spending more time on coding.			
Geras et al. [7]	2004	Academic	Professionals
Aim: To investigate developer productivity and software quality when comparing test-driven and traditional development approaches.			
Results: There were little or no differences in developer productivity but frequency of unplanned test failure was lower for test-driven development.			
Erdogmus et al. [11]	2005	Academic	Students
Aim: To evaluate functional tests in test-driven development when compared to traditional test-last approach.			
Results: Test-first students created on an average more tests and tended to be more productive. There was no significant difference in quality of produced code between two groups.			
Flohr & Schneider [12]	2006	Academic	Students
Aim: To investigate the impact of test-first compared to clasical-testing approach.			
Results: No significant differences could be established, but students did show a preference towards test-first approach.			
Janzen & Saiedian [13]	2006	Academic	Students
Aim: To examine the effects of TDD on internal quality of software design.			
Results: Positive correlation between productivity and TDD, but no differences in internal quality. Perception on TDD was more positive after the experiment.			
Müller & Höfer [14]	2007	Academic	Mixed
Aim: To investigate the conformance to TDD of professionals and novice TDD developers.			
Results: Experts complied more to the rules of TDD and produced test with higher quality.			

Janzen et al. [15]	2007	Academic	Professionals
Aim: To investigate effects of TDD on internal code quality.			
Results: Programmers' opinions on TDD improved after the experiment but internal code quality had no significant difference between test-first and test-last approach.			
Gupta & Jalote [16]	2007	Academic	Students
Aim: To evaluate the impact of TDD on designing, coding and testing when compared with traditional approach.			
Results: TDD improves productivity and reduce overall development effort. Code quality is affected by test effort regardless of the development approach in use.			
Kollanus & Isomöttönen [8]	2008	Academic	Students
Aim: To improve understanding on TDD in educational context.			
Results: Students expressed difficulties with following TDD approach and designing proper tests. Regardless, they believed in the claimed benefits of TDD.			
Höfer & Philipp [17]	2009	Academic	Mixed
Aim: To compare conformance to TDD of experts and novice programmers.			
Results: Experts refactored their code more than novice programmers, but they were also significantly slower.			
Huang & Holcombe [18]	2009	Academic	Students
Aim: To investigate the effectiveness of test-first approach compared to the traditional (test-last) development.			
Results: Test-first teams spent more time on testing than coding compared to test-last teams. There was no linear correlation between effort spent on software testing and the software external quality.			
Vu et al. [19]	2009	Academic	Students
Aim: To investigate how test-first and test-last methodologies affects internal and external quality of the software.			
Results: Test-last team was more productive and created more tests. Students indicate preference towards test-first approach.			
Madeyski [20]	2010	Academic	Students
Aim: To investigate how Test-first programming can impact branch coverage and mutation score indicator.			
Results: The benefits of the Test-first practice can be considered minor in the specific context of this experiment.			

Table 7.1: Research publications on experiments in TDD

- **Artefact Quality:**
  - $\mathbf{H}^{q}{}_{0}$. When using TDD, there is no significant difference between the quality of the artefacts produced by developers with or without knowledge in software testing.
  - $\mathbf{H}^{q}{}_{a}$. When using TDD, developers with knowledge in software testing produce artefacts of a higher quality.
- **Developer Perception:**
  - $\mathbf{H}^{p}{}_{0}$. There is no significant difference in the perception of TDD between developers with or without knowledge in software testing.
  - $\mathbf{H}^{p}{}_{a}$. Developers with knowledge in software testing have higher preference towards TDD than those without knowledge in software testing.

Construct	Variable name	Description	Scale type
Development Speed	User Stories	Number of user stories finished within lab session.	Ratio
Artefact Quality	Defects	Number of defects found in code implementation by independent test suite.	Ratio
Artefact Quality	Coverage	Statement coverage of test suite when applied to code implementation.	Ratio
Artefact Quality	Complexity	Cyclomatic complexity of the code implementation.	Ratio
Developer Perception	Ease of use	The ease of use with which the steps of TDD could be followed.	Ordinal
Developer Perception	Preference	Subjects' perception of TDD.	Ordinal

Table 7.2: Experiment Response Variables

The *development speed*, *artefact quality* and *developer perception* are operationalized in a list of response variables, provided in Table 7.2.

In this experiment, the factor of *knowledge in software testing* is operationalized using a 10-weeks half-time advanced-level academic course in Software Verification and Validiation. The course contents has been inspired partly by industrial certification courses (e.g., the ISTQB foundation- and advanced-level certification courses [21]), and partly by scientific courses and syllabi (e.g., the software testing course contents proposed by Ammann and Offutt [22]). For the purpose of this experiment, a subject is said to have knowledge in software testing if (s)he has taken part in the course lectures and exercises, and not to have knowledge in software testing otherwise. This is supported by the data collected in a survey at the beginning of the experiment where we asked students to provide information about their testing experience. Fifteen out of 22 students had no experience at all and 5 students reported to have 1-6 months of testing experience. Only 2 students claimed to have 2-3 years of testing experience.

An overview of the lecture topics of the course is provided in Table 7.3.

Lecture topic	**Time**
Introduction to software testing and testing fundamentals.	2h.
The test processes.	2h.
Workshop.	3h.
How to practically write test cases.	2h.
Code inspection and security testing.	3h.
Test design techniques.	6h.
Static program analysis.	2h.
Real-time testing.	2h.

Table 7.3: Overview of the Software Verification and Validation course contents.

7.3.2 Experiment Design

The experiment design is detailed in Figure 7.1. Two groups of subjects (Group A and Group B) worked on two different problems (Problem 1 and Problem 2) as part of the labs, one before and one after the course (using TDD on both the occasions). During both the labs they used the Eclipse [23] integrated development environment (IDE) to create software solutions in Java programming language and the jUnit [24] testing framework for writing executable tests.

Upon completion of each of the labs, the subjects answered a set of questions in an online survey system.

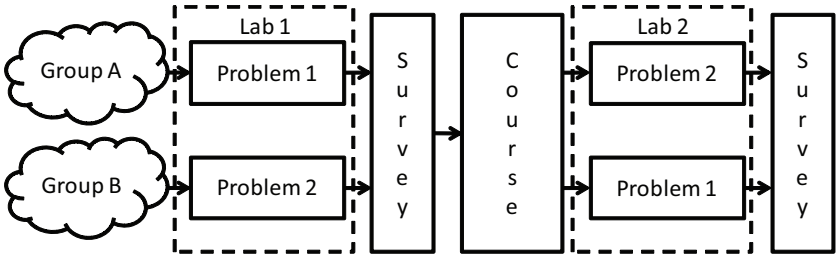

Figure 7.1: Design of Experiment

7.3.3 Subjects

The subjects of the experiment were software engineering master students enrolled in the Software Verification and Validation course at Mälardalen University during the autumn semester of 2010. The experiment was part of the laboratory work within the V&V course, and the subjects earned credits for participation. Students were informed that the final grade for the course would be obtained from the written exam and their performance during labs would not affect their grades.

7.3.4 Objects

As stated above, the experiment used two specific software development problems for the experiment, namely: (i) Roman numerals conversion (Problem 1) and (ii) bowling game score calculation (Problem 2). The specifications for Problem 1 were written by us (in the form of a list of user stories) for the purpose of this experiment, whereas the specifications for Problem 2 (also a list of user stories) were based on the Bowling Game Kata (i.e., the problem also used by Kollanus and Isomöttönen to explain TDD [8]). Detailed information about the problems and their user stories are provided on first author's webpage[2].

[2]http://www.mrtc.mdh.se/ acc01/tddexperiment/

TDD Steps:

1. Write one single test-case
2. Run this test-case. If it fails continue with step 3. If the test-case succeeds, continue with step 1.
3. Implement the minimal code to make the test-case run
4. Run the test-case again. If it fails again, continue with step 3. If the test-case succeeds, continue with step 5.
5. Refactor the implementation to achieve the simplest design possible.
6. Run the test-case again, to verify that the refactored implementation still succeeds the test-case. If it fails, continue with step 5. If the test-case succeeds, continue with step 1, if there are still requirements left in the specification.

Figure 7.2: TDD steps for development.

7.3.5 Instrumentation

As one way of ensuring that subjects properly followed the steps of TDD, we provided the instructions for TDD prescribed by Flohr and Schneider [12] (see Figure 7.2).

To avoid problems with subjects' unfamiliarity with the jUnit testing framework and/or the Eclipse IDE, subjects were given an Eclipse project code skeleton with one simple test case. Since this was all located in a subversion (SVN) repository, instructions on how to obtain code from SVN and import it in Eclipse was also provided to students.

7.3.6 Data Collection Procedure

Teams were instructed to upload their source codes in a SVN repository. This way the lab instructor had a complete log of subjects' activities and an option to obtain code from specific points in time.

The subjects answered survey questions using quiz assignments in the Blackboard[3] learning management system for the course. The data from the surveys were then exported in comma separated values (.csv) file format.

[3] www.blackboard.com

7.3.7 Validity Evaluation

During experimental design the following validity threats were identified and addressed:

1. **Maturation**
   The experiment was designed to provide different problems to subjects on different occasions in order to eliminate the factor of maturation.

2. **Problem complexity**
   By solving different problems at each instance of the lab we eliminated the issue of problem complexity.

3. **Subjects randomisation**
   Subjects were assigned into teams by the first-come first-serve method. Teams were assigned to groups in an alternating manner.

7.4 Execution

Upon defining the experimental plan and with the start of the V&V course, all pre-requirements for experiment execution were in place.

7.4.1 Sample

Twenty-eight students participated in the experiment. Since the experiment plan was to have students working in pairs, we instructed students during the introduction class to individually find a classmate that they would like to work with and send an e-mail request to the lab instructor in order to obtain a team number. Students were informed that their lab work would be used for the experiment, but they were not provided any details on the goal of the experiment itself. Also, we explicitly stated that their performance during the lab would not influence the final grade of the V&V course in any way. The final grade was determined by the written exam.

7.4.2 Preparation

Team numbers were assigned in sequential order based on the time of receipt of the e-mail requested by the lab instructor. Problems for the teams were assigned in an alternating manner between the two immediate teams (ex., if

team i was assigned problem 1, one team i+1 was assigned problem 2 and team i+2 was assigned problem 1 again etc.).

Since the lab work was time-boxed to 3 hours, a Java code skeleton was created for students. It contained a program class with one empty method returning zero and a test class with one assert statement validating the previous mentioned method. This skeleton was made to be directly imported into Eclipse as an existing project.

For each team a corresponding subversion (SVN) repository was created with read/write permissions assigned only to students within the given team and to the lab instructor. To avoid difficulties in setting up SVN and importing project in Eclipse, an instruction on the usage of SVN and Eclipse was provided to the students.

7.4.3 Data Collection Performed

As explained to students in the lab instruction document, after creating a new test or after changing code in order to pass the existing tests, a SVN commit command had to be executed. This way the lab instructor had a complete log of activities during the lab and an ability to obtain source code of the team at any given point in time. The absence of some students from any of the lab sessions were clearly visible from their SVN repository since the date of source code was not the same as the date of the lab. Such data was excluded from the analysis.

7.4.4 Validity Procedure

During the execution of the experiment the following validity threats were specifically considered:

1. Non-adherence to TDD
   We had to ensure that students are following TDD as per the instructions provided before the lab. Due to the nature of TDD it was not feasible for the lab instructor to follow in real-time if all the students were performing TDD in a correct manner. This was analysed later using SVN log and also by looking at students survey responses where they were explicitly asked to which extent did they follow the instructions for development.

2. Reuse of an existing online solution
   Internet access existed in the lab and students could potentially find complete solutions to the problems from the web. The instructor addressed

this issue by explaining to the students that (i) they have to complete the lab by solving one user story at the time and writing a specific test case for that user story, (ii) this lab work will not increase nor decrease their final grade and (iii) the strict university rules about plagiarism are relevant for the lab as well.

Based on the above measures and aspects we can confidently state that these validity threats are avoided during this study.

7.5 Analysis

The analysis section summarizes the data collected as well as the treatment of the data.

7.5.1 Descriptive Statistics

Based on initial experimental plan of response variables (see Table 7.2) a descriptive analysis was performed for each variable independently.

First, considering the development speed construct, Figure 7.3 presents percentage of user stories finished during the experiment sessions as mean values

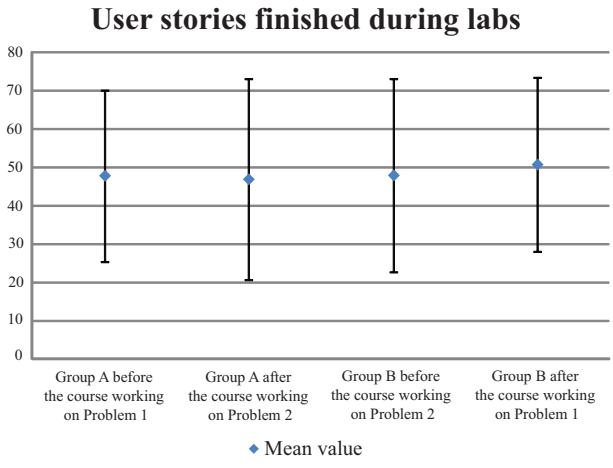

Figure 7.3: Performance mean values with error bars

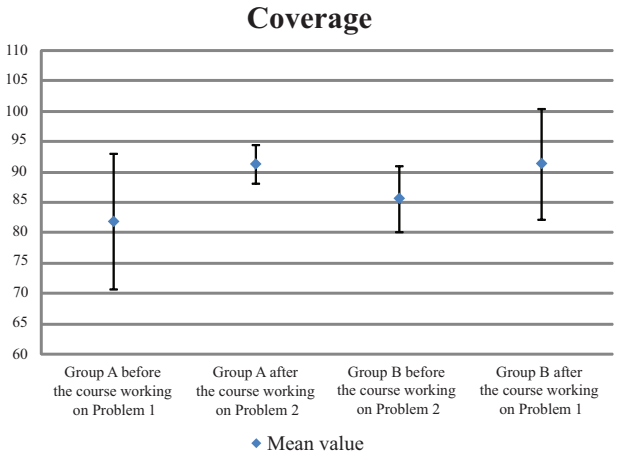

Figure 7.4: Code coverage mean values with error bars

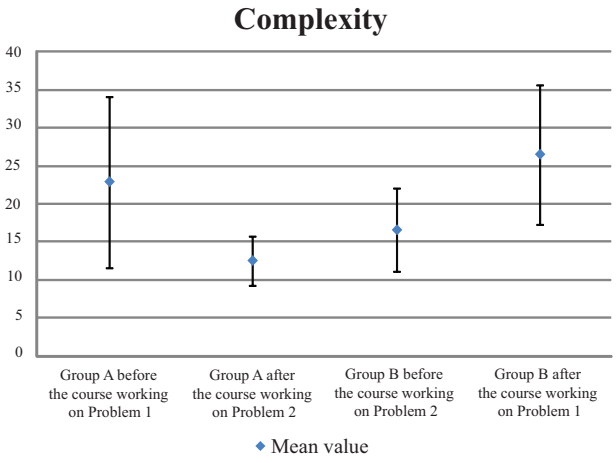

Figure 7.5: Code complexity mean values with error bars

with standard error deviation. As the figure shows, the development speed was relatively unaffected in both groups before and after the course.

Second, considering the artefact quality construct, Figures 7.4, 7.5, and 7.6 present percentage of statement coverage of students' test suites, cyclomatic

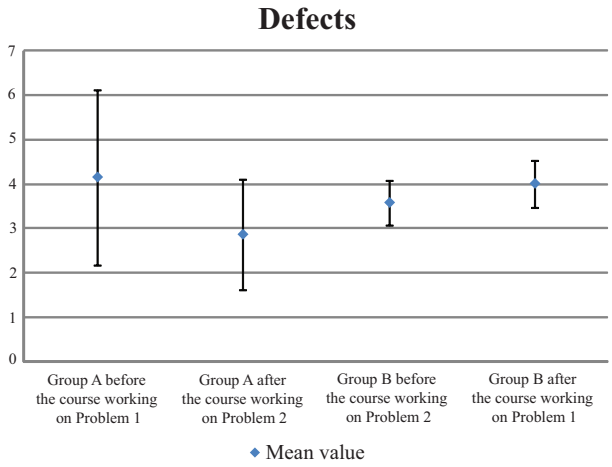

Figure 7.6: Defects found mean values with error bars

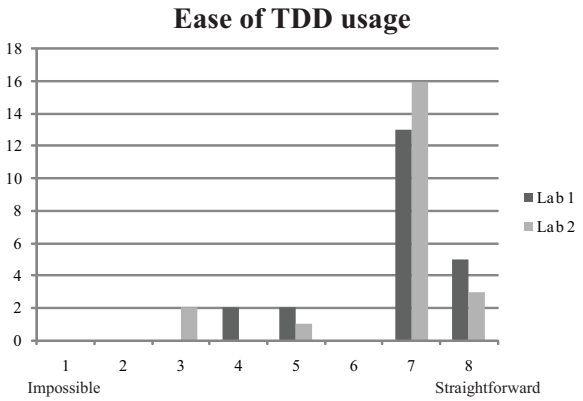

Figure 7.7: How difficult was it to follow steps for development

complexity of the code, and the number of defects detected by an independent test suite respectively. These measures are given as mean values with standard error deviations. In the case of code coverage, it can been seen that both post-test groups had better mean values than the pre-test groups. In the complexity and defects metrics, the differences between the experiment objects seem to obscure such visible results, if they exist.

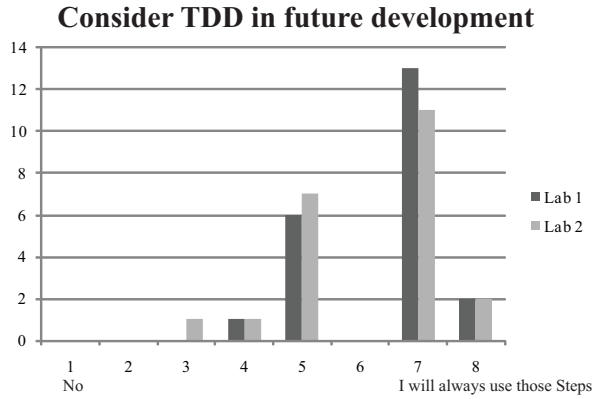

Figure 7.8: Students perception of TDD

Finally, Figures 7.7 and 7.8 provide results related to the developer perception construct. The first of these figures presents the sum of student responses on the ease of use with which the steps of TDD are followed in labs. Possible responses varies from 0 to 7 where 0 means impossible to follow and 7 means that following TDD was straightforward. Data is presented for both instances of labs. Figure 7.8 presents the sum of student responses on the perception of TDD. Possible responses vary from 0 to 7 where 0 means they will not consider using TDD in future development and 7 means they will always use TDD. Data is presented for both instances of labs. Generally, students found TDD to be a preferable development method that is easy to use. However, there is no obvious difference between the pre-experiment and post-experiment perceptions on this matter.

7.5.2 Data Set Reduction

Source codes of 17 teams (9 from Group A and 8 from Group B) and 28 student responses in survey questionnaires were collected for analysis. The difference of 6 students were due to the fact that some students did not fill in the questionnaire but did perform the lab.

When the actual source code analysis was performed additional data points had to be removed. The projects of teams 4 and 13 were excluded due to several syntax errors which made the complete solution uncompilable and irrelevant for any of the analysis. During code coverage analysis a huge deviation

occurred with Team 14. A detailed analysis revealed that students did not write any test cases during the lab but they subsequently submitted tests in SVN. Since this was opposite from the TDD practice stated in their lab instructions, data from this team was also excluded.

After removing data from those three teams, in final we had data points from:

- 14 teams (7 from Group A and 7 from Group B) for source code analysis and
- 22 student responses for survey questionnaire analysis.

7.5.3 Hypothesis Testing

Hypothesis testing was performed in two steps: First, the **Mann-Whitney** non-parametric test was used to ensure that the differences in response variable data between the experiment groups and between the experiment objects were statistically nonsignificant. The α was set to 0.05, and consequently a resulting z score of more than 1.96 or less than -1.96 was required to show a significant difference between the objects or the groups.

The result of this analysis is shown in Table 7.4. As can be seen from the table, there were no significant differences between the experiment objects or groups, with the exception of a significant difference in object complexity. This parameter is consequently omitted from further analysis.

Second, on the basis of the nonsignificant differences between experiment objects and groups, the **Wilcoxon** signed rank test for paired nonparametric data was used in order to test the null hypotheses of the experiment. As in the Mann-Whitney case, the α was set to 0.05. The result of this analysis is shown in Table 7.5. For a null hypothesis to be rejected, it is required that ***min(***W_+, W_-***)*** $\leq$ **Critical** W holds. As shown in the table, none of the experiment's null hypotheses can be rejected based on the collected data.

	Development speed	**Artefact quality**			**Developer perception**	
	User Stories	Defects	Coverage	Complexity	Ease of use	Preference
Group A vs. Group B	-0.16	-0.80	-0.34	-1.36	-0.30	1.34
Roman vs. Bowling	0.02	-1.91	0.05	**-2.64**	0.19	0.09

Table 7.4: Mann-Whitney z scores for differences between experiment groups and objects. A significant difference in complexity between the experiment objects is found.

Construct (Null hypothesis)	**Parameter**	W_+	W_-	***min(***W_+, W_-***)***	**Critical** W
Development speed ($\mathbf{H^s}_0$)	User Stories	52.5	52.5	52.5	21 (14 non-zero differences)
Artefact quality ($\mathbf{H^q}_0$)	Defects	22.5	13.5	13.5	4 (8 non-zero differences)
Artefact quality ($\mathbf{H^q}_0$)	Coverage	25	80	25	21 (14 non-zero differences)
Artefact quality ($\mathbf{H^q}_0$)	Complexity	Not tested			
Developer perception ($\mathbf{H^{pe}}_0$)	Ease of use	30	25	25	8 (10 non-zero differences)
Developer perception ($\mathbf{H^{pp}}_0$)	Preference	30	15	15	6 (9 non-zero differences)

Table 7.5: Testing of null hypotheses of the experiment using the Wilcoxon signed-rank test

7.6 Interpretation

7.6.1 Evaluation of Results and Implications

When looking at the descriptive statistics results of code coverage variable we can notice a positive increase in performances of both the groups when comparing before and after the course results. Even though there was no statistically significant differences in code coverage values (no null hypotheses could be rejected), we would like to emphasise that, on an average, the best performing group before the course was still worse than the worst group after the course.

$$\textbf{\textit{max(}}A, B\textbf{\textit{)}}_{precourse} < \textbf{\textit{min(}}A, B\textbf{\textit{)}}_{postcourse}$$

The level of complexity of the students program solutions changed for both groups from one lab to another, but this change had one direction for Group A and another for Group B. What we can only conclude from this data is that solutions for Problem 1 are of higher complexity than solutions for Problem 2.

We expected the number of defects variable to provide us with a direct way of evaluating the impact of testing knowledge. An independent suite of test cases for each problem was created but we could not use it to the full extent since different teams finished different number of user stories. This resulted that every team had on an average four bugs and in most cases those could have been found by test cases designed using negative test design technique.

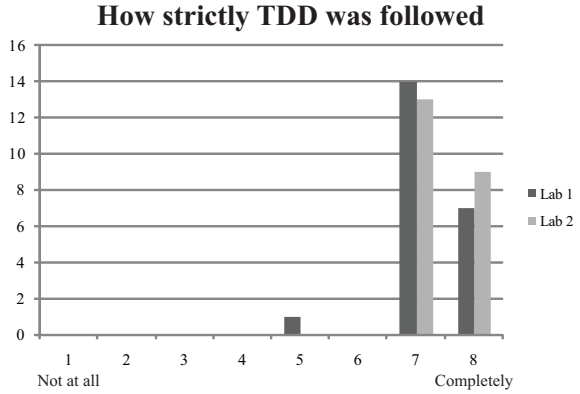

Figure 7.9: Students adherence to TDD

Students claimed they adhered to the TDD practice during the experiment to a high extent, which could be seen from the survey data presented in Figure 7.9. The ease of usage of TDD practice was also reported to a high extent (Figure 7.7) but interestingly students did not feel the same about their preference of using TDD in future development (Figure 7.8).

7.6.2 Limitations of the Study

Typically, four types of validity are discussed in empirical research (i.e., *construct validiity*, *internal validity*, *external validity* and *reliability*) [25].

Construct validity refers to the correctness in the mapping between the theoretical constructs that are to be investigated, and the actual observations of the study. Some of the constructs investigated in this study are not trivially defined, and may be subject to debate (particularly in the case of *artefact quality* and *testing knowledge*). In order to mitigate this problem, we have used standard software engineering metrics (e.g., complexity and coverage), and provided detailed information on the operationalization of each construct involved in the experiment.

Internal validity concerns the proper analysis of data. The statistical strategy used in this paper was to first eliminate the possibility of major confounding variables affecting the result (i.e., testing for differences between experiment objects or groups), and second, to test the null hypotheses. Furthermore, as the normality of the data could not be assumed, we used non-parametric tests to conduct these hypothesis tests. However, regardless of the strategy used, it is without question a fact that the sample size of the data was small, which is a major limitation for statistical analysis (and potentially also a cause for the inability for null hypothesis rejection). The only way to resolve this matter is through replications of the experiment.

External validity relates to the possibility to generalize the study results outside its scope of investigation. As many of the previously published experiments on TDD (see Table 7.1), this experiment is performed in a course setting and suffers from the consequent threats to external validity (e.g., *student subjects, small scale objects, short experiment duration*). It is, however, uncertain to what extent this affects the results, as we are not examining a practice (TDD) directly, but rather assessing whether the practice improves given the acquisition of a certain knowledge.

Reliability concerns the degree of certainty with which a replication of this study, e.g., by a different set of researchers, would yield the same study outcome. Here, as the experiment package and guidelines are made available

for replication purposes, the major reliability threat relates to the replicated execution of the V&V course. On the other hand, without having any deeper insight as to what specific testing knowledge would be beneficial for TDD, this needs to be considered for future work.

7.6.3 Lessons Learned

- **Usage of SVN repository**
  The SVN repository enabled the lab instructor to access the source code and the test cases with the proper timestamps. SVN facilitated easier management and organisation of the experimental artefacts with appropriate access rights.

- **Survey questionnaires**
  After each lab, participants answered a series of questions in an online survey system. This provided valuable qualitative insights into the experiment design and execution.

- **Lab instructions**
  By providing students with detailed instructions for the lab we minimised the possibility of misinterpretation by the students as well as the need for manual interaction with the lab instructor.

- **Code skeleton**
  By providing students a code skeleton at the beginning of the lab we eased the learning curve and the students were able to start with the experiment immediately.

- **Inability to enforce larger participation**
  We had 65 students at the start of the experiment, but due to various reasons we ended up with a data set from 28 students only. We think that the experiment would have provided more statistically significant results if we could motivate a larger participation.

- **Predefined time-boxing**
  Instead of providing fixed time duration of the lab, it would have been better for the experiment to require a specified number of user stories that had to be completed by the participants. This would have enabled us to use the same independent test suite for every solution provided by the participants in a automated manner whereas in the current setup we

had to distinguish the participants based on the number of user stories completed and manually conduct the testing.

Additional Observations

Upon execution of an independent test suite on the participants source code we noticed that 95% of the errors observed were related to unspecified behaviours. Detailed analysis of source code and test suites revealed that in both occasions (pre- and post-course) students mostly used positive test design techniques, which is referred in literature as "positive test bias" [26]. In many of these cases usage of negative test design techniques would have enabled the students to capture those errors easily.

We believe that the root cause for this observation could be the fundamental implicit orientation of TDD towards positive test design techniques since the goal is always to create the test case for a given requirement. This points towards the need for specific test design knowledge such as negative test design techniques to be provided to the developers adopting TDD in their development process together with the basic testing knowledge.

7.7 Conclusions and Future Work

In this section a summary of the study results with directions for future work are presented.

7.7.1 Relation to Existing Evidence

In related works section we mentioned three research papers where participants of their studies expressed difficulties with testing and/or constructing test cases. Opinions of the subjects of our study pointed out that testing knowledge had a relatively significant positive impact on how they performed TDD as can be seen in Figure 7.10. However, based on qualitative data from our experiment, we also inferred that our respondents had problems with creating negative test cases.

7.7.2 Impact

A growing number of research publications empirically evaluating TDD implicitly suggest that TDD will most likely provide benefit of higher code quality to the organisation which decide to implement this development process.

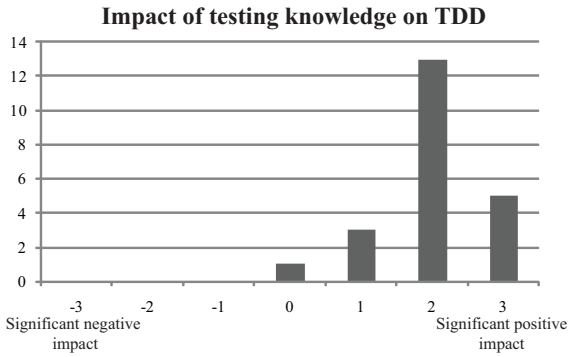

Figure 7.10: Students opinion on impact of testing knowledge on TDD

However, to the best of our knowledge, there are no reports on failure of implementing or adopting TDD within a specific organisation. In this context a more relevant research question could be: where and why TDD will not work and how to overcome those factors?

Our experiment is a initial attempt to address this research question from an orthogonal perspective by evaluating specifically whether testing knowledge can support TDD in practice or it could be considered as a limiting factor (as stated in [3]). Though the present study is inconclusive, it opens up several interesting challenges for the research community. We believe that identifying specific testing knowledge which is complementary to the testing skills of a new TDD developer that would enable developers to achieve performance efficiency and higher quality of software products, will have a great impact on the industrial adoption of TDD.

7.7.3 Future Work

In this study we presented a detailed experiment with students as subjects, making it more accessible for other researchers to replicate or perform a similar experiment. Alongside of providing more evidence on how general testing knowledge supports TDD in practice, we think an evolving experiment should be created with more specific focus. This experiment would be a possibility to directly investigate the effect of knowledge of negative testing on TDD practice. It could be designed in a way to provide education to subjects specifically

on how to design test cases for unspecified system behaviours and use that knowledge when performing TDD of software systems.

TDD per se provides an excellent opportunity for improving code quality by imbibing "test culture" in the development community. Adherence to TDD results in the generation of automated and executable test cases during the development phase itself, thus improving the testability of the system requirements. However, as indicated by our study, TDD needs to be supplemented with new process steps or test design techniques, which could potentially further enhance the robustness and the reliability of the system.

In a long term research perspective, we also intent to perform an industrial case study investigating how experienced developers could benefit from testing knowledge and what kind of specific testing knowledge they need in order to increase the quality of the code artefacts they produce.

Acknowledgments

This work was supported by the SWELL (Swedish software Verification & Validation ExceLLence) research school through VINNOVA (Swedish Governmental Agency for Innovation Systems) and the OPEN-SME EU FP7 research project.

Bibliography

[1] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[2] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. An Industrial Survey on Contemporary Aspects of Software Testing. In *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST)*, pages 393–401, 2010.

[3] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. In *Proceedings of the 4th International Conference on Software Testing, Verification and Validation (ICST)*, 2011.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[5] Andreas Jedlitschka and Dietmar Pfahl. Reporting Guidelines for Controlled Experiments in Software Engineering. In R. Jeffery et al., editor, *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005)*, pages 94–104. IEEE Computer Society, 2005.

[6] P. Sfetsos, L. Angelis, and I. Stamelos. Investigating the extreme programming system - An empirical study. *Empirical Software Engineering*, 11(2):269–301, 2006.

[7] A. Geras, M. Smith, and J. Miller. A Prototype Empirical Evaluation of Test Driven Development. In *Proceedings of the Software Metrics, 10th International Symposium*, pages 405–416, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Sami Kollanus and Ville Isomöttönen. Understanding TDD in academic environment: experiences from two experiments. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 25–31, New York, NY, USA, 2008. ACM.

[9] M.M. Müller and O. Hagner. Experiment about test-first programming. *Software, IEE Proceedings* -, 149(5):131 – 136, October 2002.

[10] Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337 – 342, 2003.

[11] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31:226–237, 2005.

[12] Thomas Flohr and Thorsten Schneider. Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings. In Jrgen Münch and Matias Vierimaa, editors, *Product-Focused Software Process Improvement*, volume 4034 of *Lecture Notes in Computer Science*, pages 305–318. Springer Berlin / Heidelberg, 2006.

[13] David S. Janzen and Hossein Saiedian. On the Influence of Test-Driven Development on Software Design. *Software Engineering Education and Training, Conference on*, pages 141–148, 2006.

[14] Matthias Müller and Andreas Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12:593–615, 2007.

[15] David S. Janzen, Clark S. Turner, and Hossein Saiedian. Empirical software engineering in industry short courses. Software Engineering Education Conference, Proceedings, pages 89–96, 2007.

[16] Atul Gupta and Pankaj Jalote. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 285–294, Washington, DC, USA, 2007. IEEE Computer Society.

[17] Andreas Höfer and Marc Philipp. An Empirical Study on the TDD Conformance of Novice and Expert Pair Programmers. In Will Aalst, John

Mylopoulos, Norman M. Sadeh, Michael J. Shaw, Clemens Szyperski, Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 33–42. Springer Berlin Heidelberg, 2009.

[18] Liang Huang and Mike Holcombe. Empirical investigation towards the effectiveness of Test First programming. *Inf. Softw. Technol.*, 51:182–194, January 2009.

[19] John Huan Vu, Niklas Frojd, Clay Shenkel-Therolf, and David S. Janzen. Evaluating Test-Driven Development in an Industry-Sponsored Capstone Project. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 229–234, Washington, DC, USA, 2009. IEEE Computer Society.

[20] Lech Madeyski. The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment. *Inf. Softw. Technol.*, 52:169–184, February 2010.

[21] The International Software Testing Qualifications Board (ISTQB). http://www.istqb.org.

[22] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, Cambridge, UK, 2008. ISBN 0-52188-038-1.

[23] Eclipse. http://www.eclipse.org.

[24] jUnit Framework. http://www.junit.org.

[25] Claes Wohlin, Per Runesson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, 2000.

[26] Laura M. Leventhal, Barbee Teasley, Diane S. Rohlman, and Keith Instone. Positive Test Bias in Software Testing Among Professionals: A Review. In *Selected papers from the Third International Conference on Human-Computer Interaction*, pages 210–218, London, UK, 1993. Springer-Verlag.

Chapter 8

Paper D:
Redefining the role of testers in organisational transition to agile methodologies

Adnan Čaušević, A.S.M. Sajeev and Sasikumar Punnekkat
In proceedings of International Conference on Software, Services & Semantic Technologies (S3T), Sofia, Bulgaria, October, 2009

Abstract

Many challenges confront companies when they change their current software development process to an agile development methodology. Those challenges could be rather difficult but one that requires considerable attention is *the integration of testing with development*. This is because in heavyweight processes, as in the traditional waterfall approach, testing is a phase often conducted by testers as part of a quality assurance team towards the end of the development cycle whereas in the agile methodology testing is part of a continuous development activity with no specific "tester" role defined.

In this paper we consider several options for testers when an organisation transit to agile methodology, and propose a new *project mentor* role for them. This role aims to utilize the knowledge that testers already have in both the business domain and the development technology together with their expertise in quality practices. This role will enhance the stature of testers as well as enable the company to effectively deploy the testers in the new environment. Motivations and benefits for this role are presented in this paper together with our plan for evaluation of this proposal.

8.1 Introduction

Software development processes have evolved over time in line with projects becoming costlier and complex. The biggest change since Royce [1] proposed the Waterfall model came with the introduction of Agile methodologies. Unlike heavyweight processes such as the Waterfall model, agile processes has encouraged customer involvement throughout the development cycle. However, initially agile processes were implemented in smaller projects with smaller teams where the risk of trying out a new process was relatively small. Increasingly, larger organisations are looking at transition of their processes to agile methodologies [2].

Testing is a prominent and continuous activity in agile processes. Paradoxically, however, testers who migrate from heavyweight processes could find their role to be diminished when their organisation implements agile processes. The reasons for this paradox include:

- the shift from testing being a high profile quality control phase to a low profile routine (daily build and test) activity
- the developers having the responsibility to test the units they build
- the need for regular interaction between developers and testers as the system gets built incrementally

Those who transition from a heavyweight process to an agile method sometimes feel that they are being micromanaged because of the constant interaction with project leaders [3]. Testers who had the role of policing the quality of the product could feel even more out of place in an agile environment, unless the transition is gradual, made with the cooperation of the testers and with adequate training. The important question to consider is not whether we need a role for testers when transitioning to an agile process but what role will they transit into.

In this paper, we study different models for the transition of a tester's role from a heavyweight process to an agile environment. We intend to test the models to assess their suitability for transitions in the real world. The rest of the paper is organised as follows. In Section 8.2 we discuss the issues involved in transition to agile with respect to the role of testers. In Section 8.3 we discuss two other approaches from the literature. Section 8.4 presents the *project mentor* model. In Section 8.5 we outline evaluation strategies to check the validity of the approach and finally Section 8.6 provides the conclusions together with the future work.

8.2 Transition to agile

Before investigating possible options for a tester or testing team organisation should define goals and parameters of transition in order to choose the appropriate option for existing testers within the company.

8.2.1 Organisational goal for transition

Organisation should clearly define its goal for the proposed transition to agile environment.

One of the possible goals could be to reduce the number of employees and often the first target is the testing team. Every employee is a valuable asset to the company and when observed as a resource testers are much more valuable than often perceived. If down-sizing is the main goal of transition then it must cover other teams as well and not only the testing department.

Most common goal for a company would be to maintain current number of employees with minimum or no investment during the transition process. The only concern for having this goal is a period of transition that could take longer than necessary. Testers will have to learn and adopt to a new process as they enter the transition. However, without any formal training or right motivation, they could have suspicion whether agile is a correct way of doing development.

Setting up an efficient transition process as an organisational goal would require to have existing number of employees in place and if possible hire additional experts. Most importantly, provide significant investment in training of the personnel. This company goal will provide employee trust in the whole process of transition and raise motivation for its successful implementation.

8.2.2 Parameters of transition

Different parameters should take into consideration when making the transition decision. Possible parameters could be:

1. To what extent: Pure agile or Hybrid?

   Some companies can adopt their development process to agile methodology only up to certain extent. On micro level, software engineering can be done in agile manner, but on a macro level things might look like developed in stages (waterfall). Example for this set-up would be in companies where product development include hardware and software parts.

2. Physical Location/Distribution of teams

   Consideration regarding physical location of the development and the testing teams is important because some options would not be possible to successfully implement if teams are distributed. Also, if one of the team itself is distributed among several locations that could create obstacles in implementing certain options for transition.

8.2.3 Options for testers during transition

Here we describe several options for the organisation regarding testers in transition to agile. For each of options we also describe their pros and cons.

1. **"Fire the testing team"**

   Process change should not start by firing existing employees. This can lead to wrong assumptions on how efficient new methodology will actually perform. Also, educating software tester and adopting to company context or even a project requires significant amount of resources. If reducing number of employees is a goal for transition process, then it should be extended to all teams within a company.

2. **"Convert them to developers"**

   Converting testers to active developers would be a reasonable option to consider, but it is not reasonable to expect testers to become developers without any formal education and specially in a short time period. Big risk with this option is a longer period to completely achieve transition.

3. **"Ask them to write test cases with developers"**

   One of the first challenges for developers transitioning to agile will be writing unit tests and understanding test driven development principles. Putting testers to work with them could be a working solution but only in a short term perspective.

4. **"Provide them with a new role Project Mentor"**

   This option represent the proposal of this paper in which we are trying to get more added value to testers in agile environment by providing them a new role of mentoring the whole project development process. This option and motivation for it are explained in Section 8.4.

8.3 Models for Transition of Testers

In this section, we are discussing two existing approaches for solving tester role while transitioning to agile environment.

8.3.1 Sumrell's approach

Sumrell [4] reports their experience in transitioning from Waterfall to Scrum. One of the major issues was to decide how to transform the QA team and their testing strategies to the new environment. The approach taken for the QA team is to continue to have the primary responsibility of testing, but share it with developers and project managers. Instead of testers waiting until the parts are ready for test, the new approach would be a quicker build cycle so that the QA team can do its work rather than having to wait. Retraining is needed for QA personnel to be able to instrument code for testing rather than rely on previous practices of automated testing strategies. However, unit testing becomes largely the responsibility of the developers.

We can identify several characteristics of this approach. One, the role of tester is somewhat diminished because some of the testing is now done by the developer. The tester requires retraining on the technical side. The tester needs to work more closely with developers and project managers thus requiring a higher level of group working skills. We hypothesise that in such an environment, a tester needs to be given adequate training for this transition, otherwise, it is likely that he or she will fail in the new environment where they are not in control of quality, and becomes just another member of a team.

8.3.2 Gregory-Crispin approach

Gregory and Crispin [5] discuss in detail the role of testers in agile development. Our model has some similarities with their approach. Their recommendation is to make testers a part of the development team. The role of testers is to help clarify customer requirements, turn them into tests, and help developers understand the customer requirements better. Testers need to speak the domain language of the customer and the technical language of the developers.

The characteristics of this approach includes, an increased role for testers as the link between customers and developers in addition to their role of testing. A shift in their work environment as they move from the Quality Assurance Division to be part of development pairs or groups. They probably will need

retraining on interpersonal skills to work closely with customers and developers more than they are used to in the past.

8.4 Our approach

We create a new role: *project mentor* in the transition from a heavyweight to an agile process. This role is different from the role of a coach which is promoted in some of the agile processes. While a coach's role is to help people adopt and implement the agile process, the role of the mentor in a project is (1) to interact with all stake holders, primarily the customers and developers and (2) to ensure that all stake holders contribute to the quality of the product under development.

Managing the expectations of customers is a difficult task in any software development project. A major task of project mentors is to manage the expectations of the customers and other stake holders. This requires domain knowledge and the ability to speak in the language of the customers, which often programmers lack. Similarly, for managers, recognising the limitations of programmers is also a difficult task. Managers without a technical background often fail to understand difficulties which are faced by programmers on a daily basis. Project mentors, we believe, will be in a position to better appreciate these difficulties and translate them to other stake holders with the help of their domain knowledge.

Agile processes try to improve quality by making quality everybody's business, not just of a quality assurance division. Testing is spread through out the development process, not just at the end of the process-chain. Agile process are sometimes called test-driven methodologies [6] for this reason. However, a drawback of this approach is that while everyone is expected to produce quality, not everyone is trained in quality assurance. A mentor's role of helping others to implement quality in their daily activities could contribute significantly to the success of the project.

We argue that the testers in a heavyweight process model are the best category of people for this new role as project mentors in an agile transition. The reasons are:

- As Gregory and Crispin [5], pointed out, testers have the domain knowledge to interact with customers as well as the technical knowledge to interact with developers. They have acquired these skills in order to implement their domain-oriented blackbox testing and the structure-oriented

whitebox testing strategies. Therefore, testers are in an ideal position to become the link between the customers and the programmers.

- Testers are trained to be quality assurance personnel. In many heavyweight process organisations, they are part of the quality assurance division. Thus it is much easier for them to transfer their quality assurance skills and mentor other personnel in inculcating the much-desired quality culture in the agile process.

We believe that there are several benefits for transforming testers as project mentors while transitioning from a heavyweight to agile process. Some of them are:

- Managers sometimes express more confidence in their testers than programmers because programmers tend to sometimes promise and not deliver (the code is 99% complete syndrome) whereas testers tell what is going wrong (i.e., the defects discovered).

- Testers are likely to become less effective or even demoralised if they are asked to be developers, because it may be difficult for them to identify themselves with this new role easily . On the other hand, an enhanced role such as project mentoring is likely to boost their morale.

- The role of project mentors which includes helping customers to write their acceptance tests and developers to write their unit tests utilises the testers' talent in an appropriate manner in the new process environment.

- Testers are no longer confined to a single location (the quality assurance division), instead they are made ”agile” and are distributed throughout the project locations, consistent with the agile philosophy.

8.4.1 Comparison of the models

Table 8.1 provides a comparison of the two existing models from literature and the “Project Mentor” model with respect to various aspects of concern testers may have while transitioning from traditional heavyweight to an agile methodology. Comparison is based from a testers perspective covering following aspects (1) Testers' stature (2) Additional skills needed (3) Responsibility and (4) Mobility.

Aspect of concern	Sumrell's experience	Gregory-Crispin approach	The project mentor model
Testers' stature	Little change	Slightly reduced	Enhanced
Additional skills needed	Both technical and people skills	Mainly people skills	Mainly people skills
Responsibility	Share with developers, project managers	Share with developers	A unique role
Mobility	Little change	Little change	Enhanced

Table 8.1: Models Comparison on Testers role from Heavyweight to Agile

8.4.2 Motivation for the new role

There are reports [7] [8] of Test Driven Development as a practice which improve quality and provide benefit to testing in general. But in order to gather testers practice and preference in particular, an industrial survey [9] on software process practices, preferences and methods is conducted. Analysing data from this survey, we found out that testers preference is highly oriented thowards incremental design, code and delivery of software. Testers are supporting frequent meetings with project members for the purpose of update on progress, but only if those meetings are planned in advance. They are also positive thowards having test cases written prior writing code. Interestingly, most of testers agrees that managers should clearly define each team members role. We think that those testers preferences are indicating high motivating reasons for including them in agile development with the specific *project mentor* role.

8.5 Evaluation plan

In order to evaluate the validity of the proposed model we have developed the following research hypotheses:

H1: Testers in current heavyweight processes have significant concerns about their transition to an agile process.

H2: Testers who have changed their role to developers when the organisation moved from a heavyweight process to an agile process were not happy to change roles.

H3: Testers favour a role of project mentors (as defined in this paper) in an agile environment in preference to a developer role or a tester role shared with developers.

H4: Managers look favourably at testers transitioning into a role of project mentors (as defined in this paper).

To test the above hypotheses there are two approaches we can take, quantitative and qualitative. Quantitative analysis will be based on a survey of a sample of the population of testers and managers. A survey instrument will be developed with items to assess testers' views on the above issues. The survey data will be statistically analysed to test for significance.

If a quantitative approach proves to be infeasible there are several qualitative solutions possible. One is the method of using case studies. In this case we will choose a limited number of organisations including ones that have already converted to agile process method and others which are considering transitioning to agile process methods. Data gathering will involve predominantly semi formal interviews with predetermined questions (with the option of asking clarifying questions).

8.6 Conclusions and future work

Agile process methodology started as a small team small project method for less riskier projects. Recently, the interest in the methodology has grown and large organisations are seriously looking at transitioning from their heavy weight processes to agile methods. One of the major challenges in the transition of personnel is how to find appropriate roles for testers when testing is not a stand-alone major phase in the development process. In this paper we have presented our views on the issue of dealing with the testing teams within a company while transitioning from a heavyweight to agile processes. We argue that it would be beneficial for the organisation to clearly define its goals and options during the transition process. We have also presented the standard options followed by transition managers together with two approaches proposed recently by researchers. We have proposed a new role called "Project mentor" for the testers in the new agile environment, and presented its advantages. In this role testers could effectively use their business domain knowledge as well as technical expertise to become the main liaison between customers and developers in order to manage their expectations and goals, as well as assist in both in writing test cases and testing the system as it evolves. We also sketched briefly our evaluation plan which we intend to take up in our future work. Our ongoing work also tries to address appropriate implementation strategies for the proposed project mentor role.

Acknowledgments

The authors would like to acknowledge the partial support provided by FLEXI-ITEA2 project[1] and PROGRESS research centre[2].

[1] http://www.flexi-itea2.org
[2] http://www.mrtc.mdh.se/progress

Bibliography

[1] Winston W. Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In *Technical Papers of Western Electronic Show and Convention (WesCon)*, 1970.

[2] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies. *Commun. ACM*, 48:72–78, 2005.

[3] Mike Cohn and Doris Ford. Introducing an Agile Process to an Organization. *Computer*, 36, June 2003.

[4] Megan Sumrell. From Waterfall to Agile - How does a QA Team Transition? In *AGILE '07: Proceedings of the AGILE 2007*, pages 291–295, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Lisa Crispin and Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009.

[6] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[7] Boby George and Laurie Williams. An initial investigation of test driven development in industry. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1135–1139, New York, NY, USA, 2003. ACM.

[8] David Janzen and Hossein Saiedian. Does Test-Driven Development Really Improve Software Design Quality? *IEEE Softw.*, 25(2):77–84, 2008.

[9] Adnan Causevic, Iva Krasteva, Rikard Land, A. S. M. Sajeev, and Daniel Sundmark. An Industrial Survey on Software Process Practices, Preferences and Methods. (ISSN 1404-3041 ISRN MDH-MRTC-233/2009-1-SE), March 2009.