TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Kirsi Korhonen

Supporting Agile Transformation with Defect Management in Large Distributed Software Development Organisation

Julkaisu 1032 • Publication 1032

Tampere 2012

Tampereen teknillinen yliopisto. Julkaisu 1032
Tampere University of Technology. Publication 1032

Kirsi Korhonen

Supporting Agile Transformation with Defect Management in Large Distributed Software Development Organisation

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 20th of April 2012, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2012

ISBN 978-952-15-2790-6 (printed)
ISBN 978-952-15-2875-0 (PDF)
ISSN 1459-2045

ABSTRACT

Korhonen, Kirsi

Supporting Agile Transformation with Defect Management in Large Distributed Software Development Organization

Keywords: agile transformation, distributed development, defect management, software quality, agile practices

Many software development organizations have started to deploy agile methods to gain improvements, for example, in production speed and quality. In a large, distributed organization the agile transformation may be a long-term, complex process. Solutions that work well in small companies turn out to be much harder to implement in thousand-developer projects. Thus evaluating achieved benefits and measuring the success of the change can be difficult, especially in the early phases of the transformation, and the lack of visibility to any progress made can decrease motivation among the personnel.

In this thesis, the general goal was to explore the impact of the agile methods when a large, distributed software organization is transforming to an agile way of developing software. The main focus of the study was to analyze the impact of agile methods on software development from defect data, and to utilize the collected data to support the organization in its agile transformation.

Data was collected with interviews, surveys, documentation analysis, and from defect data records over several years. The results show that there are changes in the defect data and defect reporting practices caused by the adoption of agile practices. Additionally, by analyzing the practices in use, by exploring the defect data, and by collecting feedback from the personnel, it was possible to provide feedback on the progress, to analyze the engagement of personnel to the transformation process, and to evaluate the success in reaching the transformation goals already at a very early stage of the agile transformation.

To conclude, this study provides research results on the impact of a large-scale agile transformation from defect management point of view in a real industrial setting. These results can be used by other organizations planning to start agile transformation to support their change process.

Preface

This journey started in autumn 2007 when I was accepted as a post-graduate student to Tampere University of Technology. I have been working in globally distributed software development projects for many years, and the topic for this thesis originates from my experiences. The journey of making this thesis in parallel to doing my regular job has lasted several years. During that time I have met and worked with numerous people who have made valuable contributions and helped me in the writing of this dissertation. I would like to express my gratitude to some of these people.

At Tampere University of Technology, my supervisor, Professor Kai Koskimies has read several versions of my publications, and helped me to revise them appropriately. He has been patient and always available with guidance when it was needed. Special thanks to Professor Tommi Mikkonen of Tampere University of Technology for reviewing the dissertation, and giving constructive feedback.

I thank Professor Michel Chaudron, Leiden University, Netherlands, and Professor Ilkka Tervonen, University of Oulu, for reviewing this thesis and contributing constructive feedback and comments on the manuscript.

Special thanks to Outi Salo for co-writing my first publication. Thanks also to Professor Pekka Abrahamsson and Professor Stephen Kan for supporting me in the writing process.

At Nokia Siemens Networks there are several colleagues to thank for making this journey possible. Erik Hiltunen has supported the study from the beginning, and Maria Lahti has been a great help in shaping the papers for publication. Other people that have helped me at Nokia Siemens Networks are Miia Forssell, Timo Vehmaro, Jari Uusinoka, Jani Eironen, Ville Pikkumaa, Timo Kulmala, Grzegorz Wanat, Waldemar Kiec, Mikko Träskilä, Suvi Ihaksi and Hanna Turunen.

I thank Nokia Foundation for funding the writing of the manuscript during the last year of this journey. The support of SoSE (Graduate School on Software Systems and Engineering) is gratefully acknowledged.

Finally, all this would not have been possible without the strong support of the closest people in my personal life. My beloved husband, Hannu Korhonen, has been the greatest support and mentor, giving me belief in my skills as a researcher and giving concrete guidance. Thanks to my lovely children Sonja and Joona for reminding what really matters in this life, and to my parents, Eeva-Liisa and Aulis, for their unconditional support in this project and for showing interest in my progress.

Thank you all for your guidance, patience and support while I was taking my steps in the area of academic research. This has been a great, fun and exciting journey, and I am glad I could share it with you. This is one dream that came true.

Kangasala, 21.02.2012 Kirsi Korhonen

Contents

Abstract i

Preface iii

Contents v

Terms and Definitions ix

List of Figures xi

List of Tables xii

PART I INTRODUCTION 1

1 Motivation 2

2 Research Problem 5

3 Research Method 7

3.1 Case study 7

3.2 Action research 8

3.3 Goal-Question-Metric 9

4 Research Process 10

4.1 General process model 10

4.2 Industrial context and data collection timeline 11

4.3 Collected data 14

5 Contributions 18

6 Organization of this Dissertation 21

PART II FOUNDATION 23

7 Agile Transformation 24

7.1 Software development 24

7.2 Agile practices 25

7.3 How to get started with agile? 27

7.4 Challenges in large-scale agile 29

7.5 Managing the change 31

8 Software Quality and Agile Development 33

8.1 Product quality standard 33

8.2 Measuring the quality 34

8.3 Defect management 35

8.4 Managing quality with defects 35

8.5 Quality effect of agile methods 37

PART III PREPARING FOR THE AGILE TRANSFORMATION 39

9 Exploring Quality Metrics to Support Defect Management Process in a Multi-site Organization 40

9.1 Introduction 40

9.2 Background 41

9.3 Research method 42

9.4 Implementation of the measurement project 42

9.5 Empirical results 46

9.6 Conclusions 48

10 Migrating Defect Management from Waterfall to Agile Software Development in a Large-Scale Multi-Site Organization 50

10.1 Introduction 50

10.2 Background 51

10.3 Research setup 53

10.4 Empirical results 54

10.5 Discussion 57

10.6 Conclusions 59

PART IV EVALUATING THE EFFECT OF AGILE TRANSFORMATION 61

11 Exploring Defect Data, Quality and Engagement during Agile Transformation at a Large Multisite Organization 62

11.1 Introduction 62

11.2 Background 63

11.3 Research setup 66

11.4 Empirical results 68

11.5 Discussion 74

11.6 Conclusions 76

12 Evaluating the Impact of Agile adoption on the Software Defect Management Practices 77

12.1 Introduction 77

12.2 Background 78

12.3 Research setup 79

12.4 Empirical results 80

12.5 Discussion 91

12.6 Conclusions 94

13 Adopting Agile Practices in Teams with No Direct Programming Responsibility 95

13.1 Introduction 95

13.2 Background 96

13.3 Research design 98
13.4 Agile experience in the organization 101
13.5 Adopting the agile practices 103
13.6 Feedback on the agile transformation 105
13.7 Discussion 107
13.8 Conclusions and future work 108
14 Evaluating the Impact of the Agile Transformation 109
14.1 Introduction 109
14.2 Background 110
14.3 Research setup 112
14.4 Empirical results 115
14.5 Evaluating the impact 123
14.6 Conclusions 125
PART V CLOSURE 127
15 Reviewing the Contributions 128
16 Limitations of This Study 130
16.1 Environment 130
16.2 Research method 131
17 Implications for Practitioners 133
18 Final Remarks 135
REFERENCES 137

Terms and Definitions

Agile practice	Software development methods associated with agile software development.
Agile software development	A group of software development methodologies based on iterative and incremental development
Agile transformation	An organization-wide change process, where agile practices are taken into use for enabling an agile software development mode of working
Case study	An empirical method aimed at investigating contemporary phenomena in their context (Runeson and Höst 2009)
Change management	A sequence of steps or activities that a change management team or project leader would follow to track, control and agree on changes in a project (e.g. Kotter 1995)
Defect	A failure in the required behavior of the system, or a fault in the software product caused by a human error (Fenton and Pfleeger 1998)
Defect (data) metrics	Quality metrics used for measuring different aspects of software-related defects
Defect management	A process that focuses on preventing defects, catching defects as early in the process as possible, and fixing defects. (QAI 1995)
Distributed organization	A multisite organization where software development is done in several physical locations by internal employees or subcontractors
Large organization	An organization which has more than 150 people working for it
Product quality	The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs (ISO 8402 (1994) standard)
Quality metrics	A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality (IEEE standard 1061)
Scrum	A framework for developing software with selected agile practices (Schwaber and Beedle 2002)
Sprint	A short, timeboxed development iteration of 2-4 weeks (Schwaber and Beedle 2002)
Waterfall model	A disciplined, sequential approach to software development

List of Figures

Figure 1. General process to support agile transformation with defect management in a large organization. 11

Figure 2. Timeline of the projects in the Studied Organization from which the data was collected in this study. 12

Figure 3. Scrum framework example. 26

Figure 4. Open defects trend line. 47

Figure 5. Defect finding profile 48

Figure 6. Defect lifetime analysis for major defects 48

Figure 7. Open faults index (% of max number of open faults) from Organizations 1, 2 and Studied organization. 57

Figure 8. Agile experience of the scrum team members based on agile transformation feedback survey responses conducted in the Organization half a year after transformation start. 66

Figure 9. Agile practices in use in the scrum teams after half a year of starting the agile transformation based on the responses on agile transformation feedback survey. 67

Figure 10. Open defects index by project during development and system testing phases. 69

Figure 11. Daily agile practices in use six months and 12 months after starting the agile transformation. 81

Figure 12. Technical agile practices in use six months and 12 months after starting the agile transformation. 81

Figure 13. Technical agile practices in use in the teams developing code 12 months after starting the agile transformation. 82

Figure 14. Survey responses to statement "*I know how to handle faults in agile software development.*" 83

Figure 15. Survey responses to statement "*New content development is more important than fixing the faults.*" 84

Figure 16. Survey responses to statement "*Our fault reporting tool is suitable for agile methods.*" 85

Figure 17. Survey responses to question if the teamwork, communication, and cooperation of teams in different locations have improved. 86

Figure 18. Open faults index. The number of open defects in projects B and C is compared to the maximum number of open defects in project A. 86

Figure 19. Defect lifetime analysis for projects A, B, and C. 87

Figure 20. Total share of critical, major and minor defects of projects A, B, and C. 88

Figure 21. Weekly number of reported minor defects over time in projects A, B, and C. 89

Figure 22. Agile experience in the Developer team (first chart on top) and SLT team (second) months (white bars) and 12 months (black bars) after starting the agile transformation.....102

Figure 23. Agile experience in the Support team 6 months (white bars) and 12 months (black bars) after starting the agile transformation.....103

Figure 24. Agile practices in use in the Developer team 6 months (white bars) and 12 months (black bars) after starting the agile transformation. First chart: Daily practices, second chart: Team practices and third chart: Programming practices.....103

Figure 25. Agile practices in use in the SLT team 6 months (white bars) and 12 months (black bars) after starting the agile transformation. First chart: Daily practices, second chart: Team practices and third chart: Programming practices.....104

Figure 26. Agile practices in use in the Support team 6 months (white bars) and 12 months (black bars) after starting the agile transformation. First chart: Daily practices, second chart: Team practices and third chart: Programming practices.....105

Figure 27. Number of open defects over time during the waterfall baseline project 1, and during the enhanced waterfall on-top projects 1a-1d.....116

Figure 28. Number of open defects over time during agile transformation in the on-top projects 2a-2b. In the background are the lines for the Project 1 and on top projects 1a-1d for reference.....117

Figure 29. Number of open defects during the baseline projects Project 1(plan based), Project 2 (plan based with enhancements) and Project 3 (agile).....117

Figure 30. Comparison of the closing times between the baseline (solid black line) and on-top releases.....120

Figure 31. Comparison of the defect closing time between baseline projects 1, 2 and 3.....120

Figure 32. Change in agile practices in the whole studied organization.....121

Figure 33. Change in the code development practices in the studied organization.....121

List of Tables

Table 1. Summary of research focus, methods and organizations involved with respect to the study periods and utilization of the data in chapters 9-14. 13

Table 2. Examples of external metrics 34

Table 3. Summary of selected metrics versus goals 45

Table 4. Main characteristics of the Studied Organizations 53

Table 5. Main results from the documentation study. 55

Table 6. Problem areas and proposed solutions. 58

Table 7. Percentage of defects found in development sprints versus system testing. 69

Table 8. Groups based on perception on code quality 70

Table 9. Mann-Whitney U test results on the number of agile practices in use between the groups. 71

Table 10. Kruskall-Wallis test results on defect questions between groups Grp1:Improved and Grp3:Gotten worse 71

Table 11. Mann-Whitney U test results for defect questions between groups Grp1:Improved and Grp3:Gotten worse 72

Table 12. Mann-Whitney U test results for defect questions between groups Grp1:Improved and Grp2:No impact. 72

Table 13. Mann-Whitney U test results for defect questions between groups Grp3:Gotten Worse and Grp2:No impact. 72

Table 14. Percentage of defects reported to own team 88

Table 15. Survey responses to the question: *Improvement in early detection of defects.* 89

Table 16. Survey responses to the question: *Reporting faults with the fault reporting tool.* . 90

Table 17. Agile transformation goals and which practices mostly support reaching the goals (marked with "x"). 113

Table 18. The results to a survey question about reporting faults with the fault reporting tool. 118

Table 19. Survey results after half a year of starting the agile transformation to the question "*what are the impacts of agile development compared to traditional development*". 122

Table 20. Recommendations for practitioners 133

Part I Introduction

This part introduces this thesis. It describes the motivation for this study, explains the research questions, introduces the main contributions and maps the research questions with the contributions. In addition, the research method and approach are described. Finally, a brief overview of the structure of this thesis is given.

1 MOTIVATION

Software development has an important role in today's business, and the usage of effective development processes within software development organizations has received great attention in the industry. According to Nerur et al. (2010), software development remains a formidable challenge: the interaction of technical complexity with the strategic nature of software development and the increased heterogeneity among stakeholders has made the context of software development more uncertain than ever before. MacCormack et al. (2003) note that choosing appropriate practices for a project can be hard, because several different software development models and practices claim to optimize various dimensions of performance.

Traditional software development is based on a sequential process approach, in which coding and testing are carried out after requirement capture and design, followed by integration and system-level testing. This plan-based approach, the so-called waterfall model, was introduced to provide a structured approach to large-scale software development (Royce 1970). In the waterfall model, project management can rely on an upfront plan, and possible deviations from the plan are easy to detect.

There are known disadvantages of traditional, plan-driven models. In large products especially, the sequential order of development has resulted in problems in integrating and testing the overall system at the end (Jones 1995). When the world around us is changing fast, requirements from the end users are changing fast as well. With the traditional software development model, it usually takes a long time from the initial requirement capture before the actual software is available for the end user, and it is difficult to implement changes in later phases. This can lead to a situation where the customers' current needs are not being addressed at the end of the project, and as a result, many of the features implemented are not used (Johnson 2002).

To be able to respond to changes faster, and to develop software systems according to the end user needs while the need still exists, many companies are searching for a solution with agile methods. The organization-wide change process, in which agile practices are taken into use to enable an agile software development mode of working, is called the agile transformation. Successful transformations from plan-based to agile software development have been reported (Schatz and Abdelshafi 2005, Petersen and Wohlin 2010), and there are frameworks designed (Sidky and Arthur 2007, Misra et al. 2006) to guide organizations in the agile transformation.

In agile software development, design, coding and testing are done simultaneously during short, time boxed timeslots (usually less than 4 weeks), which allows for fast and flexible response to change. Because customer input is available at the beginning of each time box, the software is not only delivered faster at the end, but it is also more likely to be what the end user requires. Research results suggest that agile methods, such as Test-Driven

Development (TDD) (Nagappan et al. 2008), pair programming (Begel and Nagappan 2008), refactoring (Moser et al. 2008) and Scrum (Lee and Yong 2010), help to improve product quality. These undoubtedly increase customer satisfaction with the product and with the company providing the product (Lee and Yong 2010, Ceschi et al. 2005, Sillitti et al. 2005).

Although there are clear advantages with agile methods, there are also some potential problems, such as these mentioned by Petersen and Wohlin (2010): software architecture does not usually get enough attention (McBreen 2003, Stephens and Rosenberg 2003), and agile development does not scale well (Cohen et al 2004). Additionally, the implementation of continuous testing can be difficult because of different platforms and system dependencies (Svensson and Höst 2005), and testing can even become a bottleneck with testing and development being performed in parallel (Wils et al. 2006). When the software development is also distributed to several locations, it brings new challenges as it seems to be in contradiction with one of the prerequisites of agile development – that development should preferably take place in one location so that people are physically located in the same room (Poppendieck and Poppendieck 2007). These issues need to be taken into account and managed well so that the customers' perception of the quality of the product will not be negatively impacted by potential distortions or dysfunctionality in software development during the agile transformation.

There are many views of software quality. The ISO/IEC 9126 standard defines six quality factors, including functionality, usability, and reliability. Intuitively, the occurrence of software defects (i.e. something is missing or is not correct in the software) is negatively related to functionality and reliability (Card 2004). Defects also interfere, to some degree, with other dimensions of quality. It is a fact that if defects are not resolved, it leads to customer dissatisfaction (Black 2004, Kaner et al. 1999). To manage software quality successfully by defect data, project decisions must be based on some understanding of the cause-effect relationships that drive defects at each stage of the process (Gras 2004).

Implementing agile practices will have an impact on the ways of working. Therefore one can assume that software defect management practices will be impacted as well to support the agile way of working. Thus, the software quality measurement based on defects will be impacted. For example, research results suggest that agile methods, such as Test-Driven Development (Nagappan et al. 2008), pair programming (Begel and Nagappan 2008) and XP practices (Ilieva et al. 2004) help to reduce the number of defects. One agile viewpoint (Poppendieck and Poppendieck 2007) proposes that faults are waste, and actions should be taken to fix all faults as soon as they are found to avoid faults piling up. This will be a bigger challenge with complex and large legacy systems which may already include an unknown number of bugs: these bugs do not disappear just by implementing agile practices. We can conclude that, although the number of defects has decreased, there will still be defects, both internally during the software development project, and externally reported by the customers after the software release point. Thus there needs to be a way to manage the faults formally to provide the fixes.

Even though agile practices are adopted by more and more companies, there are few longitudinal studies on the impact of agile transformation on software quality and defect management (Li et al. 2010). It is proposed that agile processes would offer better software quality (Huo et al.2004), and e.g. help reduce defect density (Ileva et al 2004, Laymann et al. 2004). Further research is needed with field studies to assess how defect management practices are affected, and how they should be revised when adopting agile methods, especially in a multisite organization.

Implementing defect management in an agile organization requires a good understanding of the agile practices and their expected impact on defects. Organizations can select suitable agile practices based on the problems they want to address. On the other hand, the availability of defect data provides an opportunity to support the organization in its agile transformation. Since an agile transformation does not take place overnight in a large organization (Vilkki 2009; Benefield 2008), lacking visibility to gradual improvements (for example, in product quality) can cause frustration and motivation decrease. This in turn may delay the agile adoption process within the organization (Benefield 2008), especially in large multisite organizations, where communication problems (Mahanti 2006; Herbsleb, et al. 2000) may further hamper the dissemination of information about improvements made. One way to provide feedback on the improvements is to compare the situation before and after the transformation by utilizing the defect data metrics, and evaluating the changes in defect management.

In this study the impact of agile transformation in a large, multisite organization has been analyzed from a defect management point of view, and the results were used to support the organization in its agile transformation. Changes in the defect data and defect management practices are presented, as well as feedback from the personnel concerning the agile transformation progress and agile practice adoption. The collected data was used to analyze the progress of the agile transformation within the studied organization, and to evaluate the impact of the agile transformation. It is expected that these results provide beneficial input for other large and distributed software development organizations that are planning to start their agile transformation.

2 RESEARCH PROBLEM

The general goal of this thesis is to explore the impact of agile methods on defect management when a large, distributed software organization is transferring towards a more agile way of making software. The main focus of the study is to analyze the impact of agile methods on software development based on defect data, and to utilize the collected data to support the organization in its agile transformation.

The detailed research questions derived from this goal are listed below.

1. Which quality metrics support the defect management process in a large, distributed organization?

In order to evaluate agile transformation impact with defect metrics, a set of metrics has to be defined that is applicable and useful in a large-scale and distributed environment. Once these metrics are available, their results can be compared before and after the transformation.

2. What kinds of challenges are there in defect management during the agile transformation?

To evaluate the results of the transformation, it is important to understand what the possible challenges are during the transformation with respect to defect management and whether there are some actions that can help the organization to prevent or overcome the challenges before they are realized.

3. What kind of changes, if any, does agile practice adoption induce in the defect data and the defect reporting practices?

According to reports (Nagappan et al. 2008; Begel and Nagappan 2008; Ilieva et al.2004), it seems logical to expect a reduction in the number of open defects. Additionally, agile practices support a faster defect cycle time since the defects are supposed to be fixed as soon as they are found. The first goal here was to study if these changes apply also to legacy code, and how fast the changes would become visible in a large, distributed organization. Another goal was to study if other changes could be seen in the defect data as a result of changing the development method.

As the adoption of agile practices usually leads to changing the ways of working, a further goal was to study how the defect reporting practices would change during the agile transformation, and what impact the changes in reporting practices might have on defect data.

4. What is the impact of visibility (or lack of visibility) to progress on the engagement to the agile transformation?

The agile transformation is a long journey especially in a large and distributed organization, where cultural differences make it even more complex to adopt new organization-wide practices. For the success of the agile transformation it is important that the people are motivated to take the new practices into use, and that they remain engaged to the change. Therefore, this thesis attempts to find evidence on whether people notice the changes in defect data and defect reporting practices, and how that affects their attitudes towards the adoption of agile practices. The research question is based on the hypothesis that those people who have a positive understanding of the impact of changes (already made) would be more engaged to the adoption of agile practices.

5. What is the impact of the agile transformation in different work roles?

To provide more background information on the agile transformation progress within the organization, and to help evaluate differences between teams, it is important to explore agile practice adaptation in teams which have no direct programming responsibility, and to compare the results with teams having programming responsibility.

6. How can the defect management data be used to provide feedback on the impact of the agile transformation?

The final research question focuses on assessing the impact of the agile transformation based on the information collected about the changes in defect data and defect management practices.

3 RESEARCH METHOD

This research was conducted as an industrial case study (Robson 2002, Yin 2003, Benbasat et al. 1987), where the impact of the agile transformation within the target organization was studied during the actual transformation by comparing the data before and after the transformation start. The data for the study was collected by surveys, interviews, literature research and observations. Action research (Lewin 1946, Susman and Evered 1978) was used in this research to study the process improvement in the organization. To define the measurement program for process improvement, the Goal-Question-Metric method (van Solingen and Berghout 1999) was used. A detailed account of applying the research methods is given in the context of each study in chapters 9-14.

3.1 Case study

This study was conducted as an industrial case study. A case study is an empirical method aimed at investigating contemporary phenomena in their context (Runeson and Höst 2009). The case study methodology is well suited for various types of software engineering research, since the objects of study are contemporary phenomena, and therefore hard to study in isolation. A large-scale agile transformation in a distributed organization is hard or impossible to simulate artificially. Thus truly reliable results concerning such environments can be obtained only in real industrial context.

A case study can be carried out in five major process steps (Runeson and Höst 2009):

1. *Case study design,* where the objectives are defined and the case study is planned.
2. *Preparation for data collection,* where the procedures and protocols for data collection are defined.
3. *Collecting evidence,* where the data collection on the studied case is executed.
4. *Analysis of collected data,* where the collected data is analyzed.
5. *Reporting threats to validity*, where the limitations of the study are reviewed and presented.

In the beginning of this study, the *case study design* was done. At first, there was one objective – to find a set of quality metrics to support the defect management in a large, distributed organization. By the time of the study, the organization had plans to start the agile transformation, and the research objective was extended to cover the impact of agile practice adoption on the defect data and on defect management. The timeline for data collection was planned to consist of two main phases: the time before the agile transformation started, and the time during the first year of the agile adoption. The data to be collected was planned to

consist of quantitative data available from defect data records, and qualitative data from interviews and surveys.

In the *preparation for data collection* phase the exact details to be collected from the defect data records were defined, the interview questions were prepared, and the survey questionnaire was created. Additionally, the participants for the interviews were invited, and it was agreed which teams would participate in the surveys half a year and one year after starting the agile transformation.

In the *collecting evidence* phase, the defect data records were collected, and interviews and surveys were conducted as planned.

The *analysis of the collected data* was done in several phases to find answers for each of the research questions as soon as the relevant data was collected. The main results were published right after the collected data had been analyzed.

The study was finally concluded with *reporting the threats to validity* phase. The limitations of this study are presented in the last chapter of this thesis.

3.2 Action research

Action research introduces a change into existing processes and observes the effects. It focuses on practical problems with theoretical relevance. In this study, the action research method was used before the actual agile transformation started for improving the defect management process, and for defining the defect-related metrics.

According to Susman and Evered (1978), action research consists of five parts: *diagnosing, action planning, action taking, evaluating* and *specifying learning.* D*iagnosing* means analyzing the current situation. The researcher's role is to interview process participants and analyze documents. In this study, the *diagnosing phase* revealed that the number of open defects had increased over time to level that was difficult to manage.

During *action planning* and *action taking* respectively, the solution is first created and then tested in practice. The researcher's role is to participate in the teamwork, and to support and observe the situation. In this study, *action planning* included defining defect-related metrics with the Goal-Question-Metric method to support project personnel in defect management, and *action taking,* respectively, implementing the selected metrics.

In the phases *evaluation* and *specifying learning* the researcher analyzes the results and describes the lessons learned. In this study, the *evaluation* of the results provided feedback for further, iterative development of the defect metrics.

3.3 Goal-Question-Metric

The Goal-Question-Metric paradigm can be used as a mechanism for defining and interpreting software measurements (Basili 1992). In this study, the Goal-Question-Metric (GQM) method, was used for providing results in the measurement program which was conducted before the agile transformation started to answer to the question of what defect metrics would support quality management in a large, distributed software development organization.

Solingen and Berghout (1999) describe four main phases in the Goal-Question-Metric method that form the timeline of a measurement program: 1) planning phase, 2) definition phase, 3) data collection phase, and 4) interpretation phase.

In the GQM *planning phase* the stakeholders of the measurement program are defined including the measurement team itself as well as the project teams which will participate in the measurement program and in the data collection particularly. The *planning phase* in this study consisted of defining what teams would participate in the measurement program, and what defect-related data would be collected to support the action research goal.

In the GQM *definition phase* the underlying goals for the measurements are defined and the selection of metrics is done by defining the appropriate questions and metrics which are expected to provide the appropriate answers. The definition phase can be further divided into three steps described by Basili (1980). The process involves first setting the goals, secondly, refining them into quantifiable questions, and thirdly, deriving the required metrics to answer the questions. In the *definition phase* of this study, the goals were identified in detail and the metrics were derived from the goals to answer the research questions.

In the GQM *data collection phase*, the availability of tools and procedures is ensured for the data collection and storage according to the plans. In this study, this phase included collecting relevant defect data records for the measurement program.

Finally, in the GQM *interpretation phase* all the metrics data from the measurement program are interpreted, while taking into consideration the specific context of these measurements. In this study, the defect metrics were analyzed to provide answers to the questions in the measurement program.

4 RESEARCH PROCESS

In this chapter, the general process model and the industrial context where the study was conducted are presented together with the data collection timeline and a description of the collected data.

4.1 General process model

Understanding and effectively leading or facilitating a change are central concerns when planning an organization-wide change. In this study, the focus was to harness defect management to support the change, agile transformation, within the organization. The main idea was to help the organization to get feedback on how the selected approach was working, and if the results were in line with the expectations.

Figure 1 presents a general process model on how defect management was used in this study to support the organization in its agile transformation. This process was inspired by the general metric program process model by Niessink and Vliet (2001).

The process is started by defining the goal (1) – what we want to achieve. In this study, the primary goal was to improve the quality of the software. The next step is to identify those agile practices (2), which support reaching the goal. Usually a change in the software development model is a long process in a large and distributed organization, and it can take years to ultimately reach the goal throughout the organization. So the problem (3) especially in this kind of environment is how do we know that the selected agile practices are taking us into the desired direction? The solution (4) in this study was to provide feedback on the changes with defect metrics – the data is usually available, easy to collect and the defect metrics are easy to understand by the people in the organization. Before the defect metrics can be used as feedback, the problem (5) on how to interpret the data needs to be solved. The solution (6) in this study was to collect feedback on how the defect management practices have changed due to agile adoption. Because the change in a large organization may take long, feedback should be collected from various sources to support the defect data analysis.

The information collected on the changes that might have an impact on the defect data interpretation is used as feedback to analyse the actual changes in the defect data (7). This analysis provides feedback on how the agile transformation is proceeding. The information can also be used as a basis for improvement actions where needed.

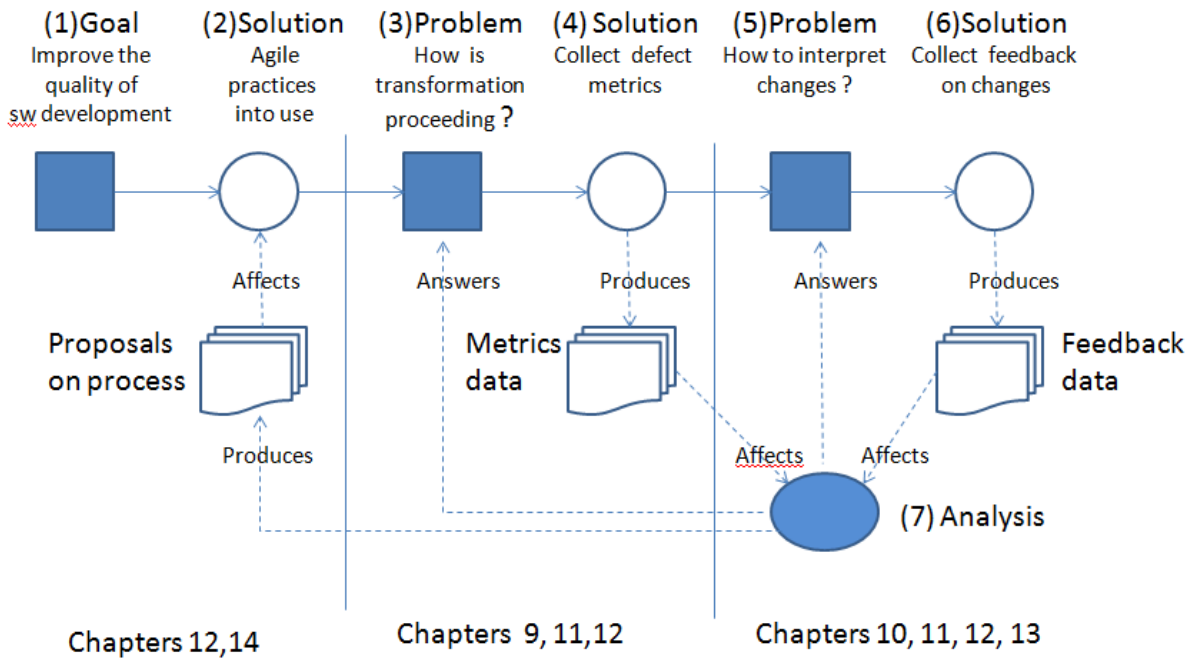

Figure 1. General process to support agile transformation with defect management in a large organization.

4.2 Industrial context and data collection timeline

The study was conducted in a large, globally distributed organization within Nokia Siemens Networks as the studied organization (later referred as Studied Organization) was transforming from traditional, plan-based software development to agile development. During the time of the research, published in articles in timeframe 2008-2011, the studied organization had over 150 people working globally in different locations across five countries in Europe and Asia. The goal of the organization was to develop the next version of an existing software product in the telecommunications domain.

Larman and Vodde (2008) define a large product group as "*one whose members' names you could not remember if you were all together in a room*". Typically this means single-product groups in the range of 100-500 people. In this study, a large product group refers to an organization with over 150 people, a medium product group to 50-150 people, and a small product group with less than 50 people in their organization.

The data collection timeline for this study is divided into three parts: plan-based period, iterative period and agile period (Figure 2). The plan-based period refers to the plan-driven approach used in the studied organization before agile transformation. The approach reflected the main characteristics of the so called waterfall model (Royce 1970). The needed functions of the software were defined first with very detailed requirements specifications. After the

design and architecture specifications were done, programming was started and finally at the end, the system level testing completed the development of the product. Changes in later phases were done through rigorous change management process. The encountered issues were related to the known disadvantages of the waterfall development model (Petersen and Wohlin 2010): long cycle time and integration problems before the system level testing.

The first step towards an agile development model was to put more focus on separate integration testing in order to find the potential problem areas in integration earlier than the regular system level testing phase at the end of the project. Therefore also the development had to be done in a more iterative manner: the whole software was integrated more often, and earlier during the development phase, to form the full system product containing those components which were ready at that time. Additional focus was put to defect management, and a defect correction prioritization practice was taken into use, with regular feedback to project personnel on the status of quality targets based on defect metrics. This period of time, which lasted for about two years, is referred to as the *iterative period* in this study.

The second step was to take more specific agile practices into use, and this was considered as the official starting point for the agile transformation. The set of agile practices was selected and implemented organization-wide. These practices included e.g. short timeboxed iterations, product backlogs, continuous integration and self-organizing teams. This period of time, which lasted for one year, is later on referred as the *agile period* in this study.

Figure 2 summarizes the timeline of the study and the software development projects from which the defect data was collected during the three data collection periods in the Studied Organization.

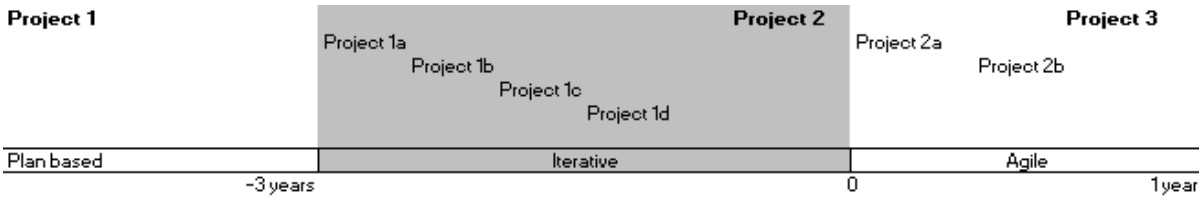

Figure 2. Timeline of the projects in the Studied Organization from which the data was collected in this study.

Project 1 denotes the baseline project with the plan-based development model. In projects 1a, 1b, 1c and 1d the software was released on top of the baseline project 1. Project 2 was the last baseline project before the official agile transformation was started. Projects 2a and 2b were developed on top of the baseline project 2 while the agile transformation was ongoing. Project 3 was the first baseline project with agile practices in use.

Before the agile transformation started in the Studied Organization, comparison data was collected from two additional software development organizations within the company and their agile software development projects: Organization 1, Project I and Organization 2, Project II.

Table 1**Table 1** summarizes the focus of study, methods used, and organizations involved with relevant projects during the three data collection periods: plan-based, iterative and agile. Table 1 also describes where the results are presented in this thesis.

Table 1. Summary of research focus, methods and organizations involved with respect to the study periods and utilization of the data in chapters 9-14.

	Plan-based	Iterative	Agile
Focus of the study	Define defect metrics to support quality management in large and distributed organization. Collect defect data from traditional development.	Identify possible problems in defect management migration to agile development methods. Collect defect data from Iterative development. Collect defect data from two additional organizations for comparison.	Analyze the impact of agile transformation in the defect data and defect reporting practices. Evaluate the success of the agile transformation.
Research method	Case study Action research GQM	Case study	Case study
Data collection method	Defect data analysis Observations by the author	Interviews Documentation study Defect data analysis	Interviews Documentation study Surveys (2) Defect data analysis
Involved organizations and projects	Studied Organization: Project 1	Studied Organization: Projects 1a, 1b, 1c, 1d, 2 Organization 1: Project I Organization 2: Project II	Studied Organization: Projects 2a, 2b, 3
Results presented in	Observations: Chapter 9 Defect data: Chapters 9, 11, 12, 14	Defect data (Studied Organization): Chapters 9, 11, 12,14 Interviews, documentation and defect data (Organizations 1 and 2): Chapter 10	Interviews and documentation: Chapter 10. Survey: Chapters 11, 12,13,14 Defect data: Chapters 10, 11, 12, 14

During the *plan-based* period, a quality measurement project was launched in the studied organization to select quality metrics that would provide visibility on the effect of the changes on defect management, and to motivate personnel in the project organization towards more efficient defect management. The goals for the measurement project were derived during action research activities and the initial defect metrics during the GQM phases. The data was collected from four consecutive software development projects, and results were compared to one project run before the changes. Later on the same metrics were utilized to analyze the changes during the agile transformation. The author participated in the measurement project and her observations on the project have been used in this study.

During the *iterative* period, data was collected from defect data records in the studied organization. In preparation for the agile transformation in the studied organization, feedback was collected from two additional organizations within the company (Organization 1 and Organization 2) on their agile transformation experiences. These two had a similar context as the organization in this study, but more experience on agile adoption. Organization 1 was large, and Organization 2 medium in size. The feedback was collected by interviews, documentation analysis and defect data records, and it was used for identifying potential problem areas in defect management during the agile transformation.

The third period, *agile,* was started with interviews and documentation reviews during the first month of the transformation to evaluate the challenges caused by the agile transformation in defect management. The defect data records were collected regularly during 12 months, and a survey was conducted twice among the people in the studied organization to make a comparison on the results after six months and after twelve months of starting the agile transformation.

4.3 Collected data

4.3.1 Defect data

The data for the defect metrics was collected from a company-wide defect tracking tool. The defect tracking tool contained a lot of information for each defect, and for this study a set of characteristics were collected for each defect. The collected data for each defect consisted of

- the title,
- the severity,
- the status,
- the date when the defect was reported,
- the date when the defect was corrected or closed as “correction not needed”,
- the date when the correction for the defect was verified if available,
- the description of the defect,
- the product the defect was found in,

- the person who reported the defect, and
- the persons who were responsible to fix the defect.

The defect data from Organizations 1 and 2 consisted only of the daily number of open defects to get the trend line over project time. This data was compared to the data collected from the studied organization when the agile transformation had just started. As these projects were from different product areas, and therefore not comparable in size or in complexity, further comparisons were left out.

4.3.2 Project documentation

To get an overview of what defect management practices and instructions were in place, project documentation was analyzed in three organizations within the company: Studied Organization, Organization 1, and Organization 2. The analyzed documentation included current versions of the project plan, project quality plan, fault related instructions and fault metric reports.

4.3.3 Observations

The author was a part of the Studied Organization, and was able to observe the situation during the measurement project and during the agile transformation. Observations were collected by participating in workshops, meetings, feedback sessions and discussions conducted within the organization. The observations were recorded as personal notes and are used in this study to provide further background information on the results.

4.3.4 Interviews

Initial information was collected on the potential challenges in defect management during the agile transformation by interviewing several key persons at the time when the agile transformation was started in the Studied Organization. The participants were selected from the Studied Organization and Organizations 1 and 2 to represent different viewpoints of the project. Participants for the interview were project manager, quality manager, system-level testing manager and a representative from a Scrum team. The project manager (referred to as PM) has the overall responsibility for the project; therefore he needs to have enough information for decision making, for example, on fault status. The quality manager (QM) defines the defect-related quality objectives and guidelines for the project organization. The system-level testing manager (SLTM) works with different teams over different sites in the organization and also needs to provide progress data to the project management. The Scrum team member (STM) has insight to the work within one team and how the defect management processes and tools support their daily activities.

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

4.3.5 Survey

While the agile transformation proceeded, the management wanted to collect feedback on the progress from the software development personnel. Questions related to defects were combined into that feedback survey. This combination of questions provided valuable feedback both on the progress of the agile transformation, and on the changes in defect management. The survey included both open-ended and closed-ended questions, offering a number of defined response choices.

The survey was conducted twice, with a six-month interval, on two main locations, as the personnel in those locations had experience both on plan-based projects and the new way of working. An additional criterion was that there were no big changes in the personnel on these two locations.

The software product developed in the studied organization is a typical telecommunications system, and consists of several bigger entities. To get a better view on different opinions the respondents for the survey were selected to represent three different functions in the software development project: software development, software verification, and software support. *Software development* personnel are responsible for implementing and verifying single components in the software system. *Software verification* personnel verify that the whole software system works as one entity once all the components have been integrated together. *Software support* personnel are responsible for maintaining environments for the software development teams and thus they support the software development and verification activities. From the first survey, altogether 96 responses were received, and from the second survey, 122 responses.

To complete the survey results, the respondents from the *software development* and s*oftware verification* functions were divided into four groups according to their perception of change in code quality. Statistical analysis was conducted to explore whether there were significant differences between the groups, and where these differences lay. Since the support personnel, responsible for the environments, were not directly involved with the software code development, these responses were left out from the statistical analysis.

Afterwards the survey results were combined, and there were feedback sessions for people in the three functions arranged by the author. In these feedback sessions the results were shared and discussion was initiated both on the reasons behind the results and for agreeing on improvement actions on areas selected during the feedback sessions. The results from these

feedback sessions and the agreed actions are not presented as such in this dissertation, but they provided useful background information for the research.

5 CONTRIBUTIONS

This dissertation makes several contributions that are elaborated in the following with a short summary.

1. Quality metrics definition for supporting defect management in a large distributed organization

This study provides a successful example of defect metrics definition and implementation to support the quality targets for a large, multisite organization. The empirical evidence shows that realistic quality targets, derived from the organization's goals and followed up with metrics, can guide defect management work and help get quality results. The measurement data supports the conclusion that the selected metrics indeed had an effect on the results. Having the criteria and following the progress regularly also changed the defect management process in the organization.

2. Identification of potential challenges in adopting agile methods for defect management in a multisite organization

This study provides a list of possible challenges in defect management migrating to agile, and proposes actions that could be taken to manage these challenges before they become actualities.

3. An approach to evaluate the progress of agile transformation in a large and distributed organization on the basis of defect data and defect reporting practices

This study provides an approach for analyzing changes in defect data and defect reporting practices, and using that information to provide feedback on the agile transformation progress. The changes in defect data metrics show the trend of whether the impact has been what was expected. An analysis on the defect reporting practices may clarify the background of the changes, and, for example, reveal improvement needs in the defect management practices.

4. Identification of the impact of progress visibility to personnel engagement levels

This study provides evidence that exposing feedback on the impact of agile methods adoption to everyone involved in the transformation can contribute to strengthening individual engagement in agile adoption, thus making the agile transformation process smoother in the organization.

5. Identification of beneficial agile practices for support and verification teams

This study reports on certain agile practices adopted in the support and verifications teams, which do not have direct programming responsibility. These practices were related to the daily organization of the work and to team work. The encountered problem areas varied between the teams, and were closely related to the work responsibility area of the team.

6. Identification of the agile transformation impact in a distributed context

This study summarizes the evidence that the impact of agile practice adoption in a large and distributed organization can be evaluated from the changes in defect data and in defect management practices. The changes, and the feedback collected from the personnel about the transformation and agile practice adoption, are used to evaluate the progress in reaching the goals set for the agile transformation by the management.

This thesis is based on the following publications:

I Korhonen K. and Salo O. (2008): Exploring Quality Metrics to Support Defect Management in Multi Site Organization – a Case Study. Proc. 19th International Symposium on Software Reliability Engineering (ISSRE 2008), Seattle, WA. IEEE Computer Society, pp. 213 – 218.

II Korhonen, K. (2009): Migrating Defect Management from Waterfall to Agile Software Development in a Large-Scale Multi-Site Organization: a Case Study. Proc. 10th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP 2009), Italy. LNBIP 31) Springer, pp. 73-82.

III Korhonen, K. (2010): Exploring Defect Data, Quality and Engagement during Agile Transformation at a Large Multisite Organization. Proc. 11th International Conference on Agile Processes and eXtreme Programming in Software Engineering XP 2010, Norway. LNBIP 48, Springer, pp. 88-102.

IV Korhonen, K. (2011): Evaluating the Impact of Agile adoption on the Software Defect Management Practices. Software Quality Professional volume 14, issue 1, December 2011.

Published also in shorter format as

Korhonen, K. (2010): Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting Practices. Proc. 7th International Conference on the Quality of Information and Communications Technology (QUATIC 2010) Portugal, IEEE Computer Society, pp. 35-43.

V Korhonen, K. (2011): Adopting Agile Practices in Teams with No Direct Programming Responsibility – a Case Study. Proc. 12th International Conference on Product-Focused Software Process Improvement (PROFES 2011). Italy. LNCS 6759, Springer, pp. 30-43.

VI Korhonen, K. (submitted 2011): Evaluating the Impact of the Agile Transformation – A Case Study. Submitted for publication in Software Quality Journal.

6 ORGANIZATION OF THIS DISSERTATION

This dissertation has been divided into four parts. The main contributions of this thesis are presented In Part III and Part IV, which are composed of the original published papers with only superficial editing. The background sections have been reviewed, and general information which is already presented in the Foundation part of this thesis has been removed. Each part of this thesis has several chapters. The following overview presents the chapters briefly to help the reader in understanding how this dissertation is constructed, and to help selective reading of this dissertation.

Part I Introduction (this part) presents an introduction to the field of distributed, agile software development, and specifically defect management, in Chapter 1. The research approach consisting of the research questions (Chapter 2), research methods (Chapter 3), and research process (Chapter 4), is presented as well. Contributions are summarized in Chapter 5, and the structure of this dissertation is described in Chapter 6 (this chapter).

Part II Foundation presents the background for the study, started with an overview of plan-based, iterative and agile software development in Chapter 7. The transformation process to agile software development in large scale is discussed next, and possible challenges in large-scale agile are presented. In Chapter 8, software quality management and defect management in particular are described with the expected effects on quality after adopting agile practices.

Part III Preparing for the Agile Transformation presents how the agile transformation was prepared in the studied organization. Chapter 9 describes the process for defining quality metrics to support the defect management process in a multisite, large organization. This chapter is based on publication I. Chapter 10 lists potential challenges in defect management when a large, distributed organization starts the transformation from waterfall software development to agile methods. Actions are proposed to overcome the challenges. The challenges and the action proposals have been collected with interviews in three different organizations. This chapter is based on publication II.

Part IV Evaluating the Effect of Agile Transformation presents how the effect of the agile transformation was evaluated in this study. Chapter 11 establishes a linkage between personnel's engagement to the agile transformation, their visibility to changes in defect data, and their perception on the impact of agile methods on the software quality. The evaluation is based on defect data and a survey conducted after six months of starting the agile transformation. This chapter is based on publication III. Chapter 12 describes the impact of the agile practice adoption on defect data and defect reporting practices. Information has been

collected from defect data records and a survey after six months of starting the agile transformation. This chapter is based on publication IV. Chapter 13 describes how three different groups of people in the software development organization implemented the agile practices, and what challenges they faced. People in two of these groups do not have direct programming responsibility. This chapter is based on publication V. Chapter 14 summarizes the study results to evaluate the impact of the agile transformation in a distributed organization, based on the collected data and evaluated against goals defined for the transformation by the management. This chapter is based on publication VI.

Part V Closure concludes this thesis. Chapter 15 reviews the contributions, and the limitations of the study are discussed in Chapter 16. Chapter 17 discusses lessons learned from this study for practitioners. This part is concluded with final remarks and future study proposals in Chapter 18.

Related work has been presented and discussed in Chapters 9-14 with respect to the research questions in each chapter. Recent related work, which has been published after the original publications of this thesis by October 2011, has been presented in Part II mostly in sections 7.3, 7.4 and 8.4.

Part II Foundation

This part presents the general background for this thesis by introducing the characteristics of waterfall and agile software development methods and describing the basic concepts of software quality management and defect management. A more detailed background is given in the context of each study in Chapters 9-14.

7 AGILE TRANSFORMATION

In this chapter, the software development paradigms, and the practices related to agile development in particular, are described in brief. The chapter is concluded with examples of the agile transformation process and potential challenges in large-scale agile transformations.

7.1 Software development

In response to the problems of managing large software development projects, the plan driven development model was introduced in the late 1960s. The aim was to bring control and discipline to what had previously been a rather unstructured and chaotic process (MacCormack et al. 2003) by planning in detail the whole software development project in the beginning. Leffingwel (2007) agrees that, despite the nowadays known problems, *"the model provided a logical way to approach building complex systems, and it provided time for the up-front requirements discovery, analysis, and design process*".

The article by Winston Royce (1970) is often cited as the first formal description of the plan driven model called waterfall - though Royce presented this model as an example of a non-working model requiring additional features to eliminate most of the development risks. In his article, Royce (1970) lists seven sequential steps which form the software development process: system requirements, software requirements, analysis, program design, coding, testing and operations. In practice, in the plan driven development the requirements are defined in a very detailed level in the beginning, followed up by design specification before implementation can be started, and verification is done after implementation is completed. According to Royce, this is risky and invites failure, because the testing is done late in the process, and thus the difference between the outcome and what was expected during planning is found very late, which causes extra effort to modify the requirements or to change the design.

Petersen and Wohlin (2010) also conclude in their more recent analysis that in plan-driven models, the main cause for failure is related to poor requirement handling. This has resulted in problems with integrating the overall system and testing it at the end (Jones 1995). Making any changes in later phases is very difficult, and requires rigorous change process implementation. This can lead to a situation where customers' current needs are not addressed by the end of the project (Leffingwell 2007), resulting in that many of the features implemented are not used (Johnson 2002).

The requirement for a pre-defined plan in plan-driven development together with the sequential order of software development have often resulted in documentation-driven, heavyweight software development processes. The *Agile Manifesto* was created in 2001 to

respond to the need for a more flexible software development model. In the Agile Manifesto (2001) it reads:

"*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- *Individuals and interactions over processes and tools.*
- *Working software over comprehensive documentation.*
- *Customer collaboration over contract negotiation.*
- *Responding to change over following the plan.*

That is, while there is value in the items on the right, we value the items on the left more."

Agile development does not emphasize up-front plans or strict control over the plans. Instead, it relies on people and their creativity rather than on formalized processes (Cockburn and Highsmith 2001). It also provides a mechanism for change management (Nerur and Balijepally 2007). In agile software development, the key thing is to respond to customer needs fast and with good quality. In practice the target is to have flexible requirement and change management in addition to short release cycle. According to Leffingwell (2007), in agile, "*we do not assume that we, or our customers can fully understand all the requirements, or that anyone can possibly understand them all up front*". There will be changes constantly (Leffingwell 2007), thus we need to work in short (max 4 weeks), timeboxed iterations. To assure that there is always a product available for demonstration or potential shipment, continuous integration needs to be in place (Leffingwell 2007, Poppendieck and Poppendieck 2007). Finally, to guarantee that the customer receives the most important features earlier rather than later, immediate and direct customer feedback is needed (Poppendieck and Poppendieck 2007, Leffingwell 2007).

Although planning is to be done for short iterations only, there needs to be an overall release project plan especially in large scale products to align activities between the teams. The difference with the plan-driven approach is that in agile the full content of the release is not fixed in the beginning of the project for the whole product, and the planning is done in detail only in the beginning of each iteration with respect to the latest understanding of the requirements for the software.

7.2 Agile practices

There are several different practices, which are often mentioned when discussing agile software development. The agile implementation within one organization can be a combination of many but not necessarily all of these. Among the most commonly adopted and known practices are *Scrum* (Schwaber and Beedle 2002) and *Extreme Programming (XP*) (Beck 2000). In this section the practices used in the studied organization are discussed in brief.

According to Schwaber and Beedle (2002), Scrum provides a framework for developing software in agile (Figure 3) and it focuses on project management. The items to be developed are collected by *Product owner* from the *Customer*, other *Stakeholders*, and from the *Scrum team*, a cross-functional and co-located team of 7 (+/-2) people. The items are added to a *Product backlog*, which is an ordered and prioritized list of work items to be developed.

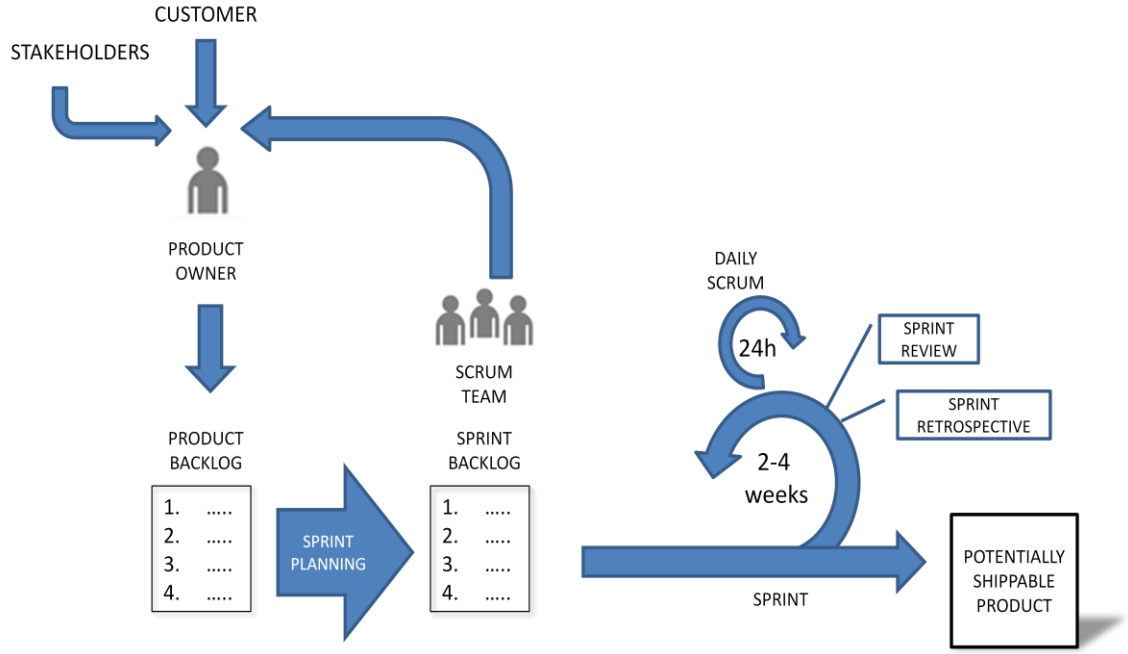

Figure 3. Scrum framework example.

The work items in the backlog are described as *user stories*, which cover the requirements and exception conditions. The *Sprint*, a short, timeboxed period of 2-4 weeks, is started with a *Sprint planning meeting*, where the *Scrum team* will plan the work for the sprint according to the Product backlog and guidance provided by the Product owner. Scrum team will take the first user stories from the Product backlog to their *Sprint backlog*, a work list of items to be developed during the next Sprint. The Scrum team is *self-organized*, which means that they have the authority and responsibility to decide which tasks they can manage during the sprint, and what working methods to use. The relative work effort needed for each user story can be evaluated with *Planning poker* (Cohn 2005) during the sprint planning session. The sprint is ended with a *Sprint review*, where the Scrum team will present the results of their development work to the Product owner. The Scrum team will also arrange a *Sprint Retrospective* meeting to evaluate and improve their working methods for the next sprint.

As new functions are being developed in each sprint by Scrum teams, integrating the new code from several developers with the existing code base needs attention and change management. *Continuous integration* (Duvall 2007) aims to improve the software quality, and to reduce the risk of unwanted changes in the code, by revealing the bugs at an early phase. The idea is that developers integrate their new code with the existing code regularly in short intervals to avoid too many changes between the product main line code and the

baseline they used for developing their new code. The integrated new build can be triggered manually by the developer, or automatically. Test automation is preferred especially in large products, and it helps to find bugs faster. One disadvantage in continuous integration is that it takes time to set up the system.

In agile development, testing is not a separate phase but done continuously to make sure that the design is good and user stories are fully implemented during the sprint. *Test-driven Development* (TDD) (Beck 2003) is a software development method where the developer first writes a test case for the user story to be implemented. Obviously, first the test case should fail as the feature has not been implemented yet. After the failing test case has been done, developer(s) will write the code for the new functionality, and then run the same test case again to see if the desired improvement or new function has been acquired. The test case is run also later on automatically to make sure that the newly developed code or refactoring does not break the existing functionality. *Refactoring* means improving the design of the existing code (Fowler 1999). According to Fowler (1999), refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. By refactoring, the code becomes easier to read, to work with, and to maintain (Kerievsky 2004). *Acceptance Test Driven Development* (*ATDD*) is a similar method to TDD. Elisabeth Hendrickson (2008) defines ATDD as a practice in which the whole team collaboratively discusses acceptance criteria, with examples, and then distils them into a set of concrete acceptance tests before development begins.

Collective code ownership, *sustainable pace* and *pair programming* are three of the 12 practices, which are listed to form an agile software development method called *Extreme programming* (XP) (Beck 2000). XP focuses on the implementation of software. In pair programming, two people are working on the same workstation. One is doing the coding in more detail while the other thinks about the big picture. This is a good way to share information within the team, and also learn new areas from a senior developer. Pair programming supports the collective code ownership, where everyone is responsible for the code and is allowed to change any part of the code. All practices listed above aim to have well-tested, continuously integrated, regularly released code with good quality. As development is done in short cycles, there should not be a need for last-minute overtime work before the release date, but people are working at a *sustainable pace*, i.e. not more than 40 hours per week over the project.

7.3 How to get started with agile?

Agile transformation introduces a change, and it is a challenge to any organization (Lawrence and Yslas 2006, Cohn and Ford 2003, Lindvall et al. 2004). Initial experiences with agile practices can be tough, for example Scrum can put more stress and time pressure for developers (Li et al. 2010) and there can be serious personnel-related challenges, including decreased motivation and fear of increased social interaction (Conboy et al 2011). As the

problems become painfully visible, changes are required in the working habits and mindsets of individuals, teams and organizations alike (Ranganath 2011). Despite the known problems, the success stories of agile methods (Schatz and Abdelshaft 2005, Lindvall et al. 2004, Lifshitz et al. 2008, Ileva et al. 2004, Laymann et al. 2004), agile in distributed context (Lee and Yong 2010) and agile in large organizations (Petersen and Wohlin 2010), have encouraged other organizations to start the journey towards an agile way of working.

The first, general guideline on how to get started with agile development is to start using some of the agile practices, and while learning more, enhancing the working methods. This may work as the first step for an individual - a developer can easily start refactoring his code, or writing test cases for the code as in TDD. But getting the whole organization take up agile practices, for example, start using a common product backlog or sprint schedule, may require a more structured approach especially in a large organization.

According to Qumer and Henderson-Sellers (2008), very few organizations are psychologically or technically able to take on an agile approach rapidly and effectively. Rohunen et al. (2010) point out that the problem often comes when all agile values, principles and practices have to be combined and implemented in practice. The agile adoption process depends on the organizational environment including staff personality, management style and cultural issues. Thus tailoring is often needed to integrate the agile practices and methods to existing processes (Rohunen et al. 2010).

A number of approaches are available to support organizations in their agile transformation. According to Sidky and Arthur (2007), any structured approach to guide organization in their transformation should include at least determining four issues: 1) the organization's readiness for agility, 2) the practices it should adopt, 3) the potential difficulties in adopting them and 4) the necessary organizational preparations for the adoption of agile practices. To address these requirements, Sidky and Arthur (2007) introduce a framework for estimating the agile potential in an organization, and for determining which agile practices can be introduced into the organization, and to what extent. This framework does not depend on any particular agile practice or method. In a similar agile framework, which is tailored specifically for mission- and life-critical systems, Sidky, Arthur and Bohner (2007) emphasize that some agile practices depend on the presence of other practices during their adoption. For example, continuous integration requires automated unit tests, and self-organizing teams require motivated and empowered individuals. This is valid also for systems which are not life-critical.

Qumer et al. (2007) introduce an Agile Adoption and Improvement Model (AAIM) to support organizations in adopting, assessing and improving their agile software development model. AAIM is a method-independent model. To further assist managers to assess the required degree of agility, and how to identify the appropriate ways to introduce agility in their organization, Qumer and Henderson-Sellers (2008) introduce Agile Software Solution Framework (ASSF). It includes an Agile Toolkit, and an overall setting for exploring the agile methods, knowledge and governance. Together these elements of ASSF support in aligning the business value and the agile process. Similarly, a recent study by Tseng and Lin

(2011) has a business aspect, and they provide a structured procedure for agility in operational strategy planning.

Based on industrial experience, it seems that a gradual implementation of agile practices works better than a strategy in which the entire agile process is adopted at once (Hodgetts 2004). Also Kent Beck (2000), the founder of Extreme Programming, proposes to adopt one practice at a time which always addresses the most pressing problem for the team. In large, distributed organizations communication may be an issue, and thus, one approach could be to start with practices which are evaluated to be most beneficial for team coordination, or for communication (Pikkarainen et al. 2008).

Who should then make the decision on which practices to implement? In an agile adoption study based on literature examples and experiences from industry, Rohunen et al. (2010) conclude that both bottom-up and top-down approaches have their roles in agile adoption, especially in large organizations. To empower the teams, and to keep them self-organized, teams should be allowed to choose and adopt their agile practices. On the other hand, actions are needed also on the management level to align the business activities, management culture and organizational values, for example, implementing a product backlog (Rohunen et al. 2010).

7.4 Challenges in large-scale agile

Usually agile practices and lean thinking are proposed to be used by small, co-located teams. Since agile methods are getting more popular, they "*are fast becoming the adopted development methodology commercially"* (Tan and Teo 2007) and agile methods are introduced also into software companies that have distributed teams around the globe, and where the organization size is large. According to Conboy et al. (2011) this presents new personnel and human-resource-management challenges, since the agile methods are applied in environments outside their original scope. They argue that adopting agile methods is no longer an insular, bottom-up, voluntary decision done by the team as originally done in smaller organizations. This creates a need to pay more attention to associated challenges with skills and people during the agile transformation process.

According to Lee and Yong (2010), most challenges that distributed teams face are related to communication and trust. Especially in a large, multisite organization additional challenges may be caused by dependencies between teams on different sites (Mahanti 2006), and a high number of teams may require support in coordinating and making decisions related to planning timelines for concurrent projects. Further challenges have been reported, concerning, for example, communication (Conboy et al. 2011, Mahanti 2006, Herbsleb et al. 2000), coordinating testing, and increased test coverage and integration of release projects in the overall development process (Petersen and Wohlin 2010).

In that kind of situation some of the agile practices need scaling for improving the support to the needs of the large organization. The agile methods originally focus on reacting to the changes and new requirements, but in large organizations developing complex products, the perspective has to be extended from individual release projects to long-term portfolio management and product evolution (Kettunen and Laanti 2007). It affects also budgeting, beta-testing, launch and governance, and HR policies (Larman and Vodde 2008) - emphasis on simple and enjoyable tools is especially important when working in large-scale product development. The ability to push practices and processes grows weaker as the group size increases (Larman and Vodde 2008). A study made at Nokia Siemens Networks (Vilkki 2009) concludes that several agile engineering practices as well as a sustainable pace need to be in place before significant improvements on code quality can be reached. Therefore the quality effect cannot be expected to become visible immediately, as in large organizations getting to a fully agile level can take a few years (Rohunen et al. 2010). A lack of visibility to the improvements made can cause frustration in the organization, and result in resistance to the change.

Even though there are clear advantages with agile methods, there are also some known issues, as listed by Petersen and Wohlin (2010): architecture usually does not have enough attention (McBreen 2003, Stephens and Rosenberg 2003), and the implementation of continuous testing is difficult because of different platforms and system dependencies (Svensson and Höst 2005). Testing can even become a bottleneck, because testing and development need to be ongoing at the same time (Wils et al. 2006). Abrahamsson et al. (2010) conclude in their review on agile methods that most of the methods fail to provide adequate project management support, and after ten years of agile adoption, the empirical evidence on its benefits is still quite limited.

The emergence of self-organizing and empowered teams, with increasing expertise and motivation, is proposed to be one of the main success factors in agile software development (Chow and Cao 2008). Indeed, it is very difficult for team members to commit to a change initiative if they do not fully understand their responsibilities (Rangananth 2011). Providing visibility to progress reinforces motivation and empowers the team during the agile transformation (Poppendieck and Poppendieck 2007). But measuring one's success against a predetermined plan can be difficult, because in agile development the plan can change in every sprint. One solution can be to measure the product quality and time to market instead of comparing to the original plan (Schatz and Abdelshafi 2005), or to share experience reports on successful agile projects with project teams (Conboy et al. 2011). This is especially important in a large organization, where the physical distance between the teams as well as cultural differences may cause additional challenges in adopting the practices and measuring the progress.

Global, large-scale software development often involves subcontracting. Using agile practices in this kind of environment may be challenging as each company is expected to look after its own interests, and the contract defines the rules for the co-operation (Poppendieck and Poppendieck 2007). In this case, contracts that avoid fixing the scope in

detail (for example time-and-material contracts) can work better in agile software development.

A mature, agile organization looks at the whole system, and does not focus on optimizing disaggregated parts (Poppendieck and Poppendieck 2007). The ability of the system to achieve its purpose depends on how well the parts work together, and not just how well they perform individually. According to Poppendieck and Poppendieck (2007), this calls for systems thinking. First, instead of pushing for growth, the organization should find and remove the limits to growth. Secondly, people should always seek for the underlying root cause instead of addressing the symptoms of an underlying problem. Finally, the more complex the system, the more tempting it is to divide the system into smaller parts and manage the parts locally. This usually leads to local measurement of performance, which may have system-wide effects that decrease the overall performance.

In a recent study by Hannay and Benestad (2010), potential threats to productivity were studied in a large agile development project. They made similar conclusions on issues which have been presented in this section: they can be threats to productivity. They further propose that there are favorable conditions: high skills, anchored methods, co-location and user involvement, which can moderate the effect on productivity in a large-scale environment.

7.5 Managing the change

Agile transformation process is a change that includes additional challenges in a large and distributed organization. In literature, several change models have been proposed to guide organizations on implementing a major change. For example, Kotter (1995) introduced an eight-step strategic model for transforming organizations. According to Kotter (1995), "*leaders who successfully transform businesses do eight things right (and they do them in the right order)*". In his model the change is described as series of phases, and a mistake in any of the phases can lead to a delay in the change process. Mento et al. (2002) add to the list in their 10-step model, for example, that the target group of people need to be prepared for the change, and lessons learned should be integrated to the change process. Common for the models is creating in the beginning a clear, shared vision for the change, a vision that can be understood by the people in the organization. Additionally, when the change can continue over months or even years in a large-scale agile transformation, it is necessary to consider the progress made: is the change still going into the right direction, or are corrective actions needed (Mento et al. 2002).

To provide feedback on the transformation, and on the performance of the organization, a measurement program can be implemented. The measurement program provides metrics that can be used to support the organization in its change process. A general process model as described by Niessink and Vliet (2001) includes four steps:

1. Improvement analysis, where the organizational problem is analyzed.
2. Measurement implementation, where the possible causes and solutions to the problem are defined. Data collection is also established.
3. Measurement analysis, where data is gathered and analyzed with respect to the proposed causes and solutions.
4. Improvement implementation, where the organization implements the solution to solve the identified problem.

According to Niessik and Vliet (2001), a measurement program can be successful within its larger organizational context only if it generates value for the organization. Hall and Fenton (1997) conclude in their study on implementing effective software metrics program that "*the success of a metrics program depends upon a carefully planned implementation strategy*". Practitioners need to be involved in the metrics program design, and the progress of the program needs to be available for them. Additionally, data collection should be automated wherever possible, and the implementation should be incremental.

8 SOFTWARE QUALITY AND AGILE DEVELOPMENT

In this chapter, first the standard definition for product quality is presented with discussion on how to measure the quality. Defect management as part of quality assurance is described next. The chapter is concluded with discussion on the expected quality effect of implementing agile practices.

8.1 Product quality standard

In the ISO 8402 (1994) standard, product quality is defined as "*the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs*". Defining and measuring software product quality is a complex but critical success factor in software development. According to Card (2003), software quality is a key dimension of project performance, equal to cost (effort) and schedule. To keep customers satisfied, the organization needs to meet their quality requirements. The ISO 9000 family of standards "*represents an international consensus on good quality management practices*". It consists of quality management systems standards and guidelines. In this thesis, a well-known industry standard ISO/IEC 9126 from the ISO 9000 family is used to further define product quality.

The ISO/IEC 9126 standard is targeted to ensure the quality of all software products, dividing quality into external and internal quality. External quality means "*the extent to which a product satisfies stated and implied needs when used under specified conditions"* (ISO/IEC 9126). Measurement of external quality is relevant when running and executing the software. Internal quality is defined as "*the totality of attributes of a product that determine its ability to satisfy stated and implied needs when used under specified conditions*" (ISO/IEC 9126). It is possible to measure internal quality by measuring the software itself while software execution is not required.

The ISO/IEC 9126 standard provides both quality characteristics and associated metrics for organizations to define a quality model for a software product. The external and internal product quality are specified with six characteristics: functionality, reliability, usability, efficiency, maintainability and portability (ISO/IEC 9126). From these, reliability is related to software defects, and maintainability to correcting the defects.

A standard on Software product Quality Requirements and Evaluation (SQuaRE) ISO/IEC 25000:2005 contains terms and definitions, reference models, general guide, individual division guides, and standards for requirements specification, planning and management, measurement and evaluation purposes.

8.2 Measuring the quality

According to the IEEE standard 1061, a metric is a measurement function, and a software quality metric is "a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality." To develop a set of metrics for a project, a list of quality factors that are important for the project needs to be created (Kaner 2004). For each quality factor there should be a direct metric that serves as a quantitative representation of the quality factor.

Defining quality criteria and related defect metrics can provide support for decision making in the organization (Basili 1992), thus the quality criteria definition should be done carefully. But some of the quality factors are difficult to measure directly (Card 2003). Intuitively, the presence of defects makes functionality and reliability worse – but how to put that into metrics? Additionally, team member participation is considered important in achieving quality results, and management should direct the organization and individuals towards common goals (Soin 1992). Therefore the quality criteria should be motivating and realistic, so that people can commit to them, and they should be formulated in a way that everyone can understand them in the same way.

It has been suggested that there should be 5-7 critical metrics to measure the product quality, and out of those, 2-3 would be related to defects (QAI research report 1995). Examples of internal metrics to measure the quality characteristics are given in the ISO/IEC 9126 part 3 standard, and examples of external metrics are listed in ISO/IEC 9126 part 2 (Table 2).

Table 2. Examples of external metrics

Name of the metric	Explanation
Failure density against test cases	How many failures were detected during defined trial period?
Failure resolution	How many failure conditions are resolved?
Fault density	How many faults were detected during defined trial period?
Fault removal	How many faults have been corrected?
Mean time between failures	How frequently does the software fail in operation?

8.3 Defect management

Various defect management methods are used by software developers and testers. The basic defect management process includes defect prevention, defect discovery and resolution, defect causal analysis and process improvement (Jäntti 2006). There are also other models available (Mays et al. 1990, Florac 1992) and practical instructions for establishing a defect management process in an organization (Jäntti 2006) and in an agile development process (Kokkola 2004). There are examples from industry (Mays et al. 1990, Daskalantonakis 1992) showing that improving the defect management process improves software quality. Descriptions of defect prevention processes (Mays et al. 1990) and fault reduction methods (Boehm and Basili 2001) suggest improvement possibilities for the defect management process.

The main goal of defect management is to increase the software quality by finding and fixing defects as early as possible. According to Fenton and Pfleeger (1998), the longer a defect stays open in the system, the more irrelevant it becomes. They further describe software defects to be both failures in the required behaviour of the system, and faults in the software product caused by a human error.

In order to systematically investigate the defects and to provide corrections for them, information on each defect must be recorded in a reporting tool (Fenton and Pfleeger 1998). The problem report of the defect usually contains elements needed for measurement purposes, such as timing and severity (Fenton and Pfleeger 1998). Timing defines the lifetime of the defect from the moment when it was detected and reported to the point when it was corrected and verified. Severity describes how serious the failure's consequence was for the service required from the system or how big an impact the fault potentially imposed on the end user. The severity scale, such as "*Critical*", "*Major*" and "*Minor*" (Fenton and Pfleeger 1998), should be well-defined.

8.4 Managing quality with defects

Typically the most important aspect of quality is correctness, i.e. how well software conforms to requirements and specifications (Tian 2004). Each phase of the software development – e.g. design, implementation, system integration - has some ability to insert and detect defects. If the project processes remain stable, the defect insertion and detection rates tend to remain relatively constant or vary within a recognized range (Card 2003).

Gras (2004) suggests that a defect model, which describes the factors that drive defect occurrence and removal in the software product life cycle, enables successful software quality management by data. Successful decision-making requires understanding the cause-effect relationships that drive defects' occurrence at each stage of the process.

According to Chillarege et al (1992), there are two main streams of how defects can be utilized to provide feedback on software development. First, statistical defect modelling (for example defect detection rate, number of defects remaining in the field) provides a good report for software reliability analysis. On the other hand, the information usually comes so late in the software development process that it is not useful for developers as a feedback channel. Causal analysis provides a second option for feedback to improve the software quality. In causal analysis, the root cause of defects is identified, and actions are initiated accordingly to eliminate the source of defects. This method gives direct and timely feedback for software developers, but especially in large products this requires significant resources to administer. Chillarege et al. (1992) present a third option in between these two extremes called Orthogonal Defect Classification (ODC). In ODC, defects are categorized into classes that collectively point to the part of the process that requires attention. For example, defect types (describing what is missing or incorrect in a function, interface etc.) can be used to provide feedback on the development process, and defect triggers, a condition that allows defect to surface, can provide feedback on the verification process.

Card (2003) describes in his article two statistical defect modelling approaches. In both approaches, a life-cycle defect profile, "quality budget", is developed first. During project execution, expected defect levels are compared to actual defect levels as the project progresses. Any deviations from the plan should be investigated, and corrective actions can be done accordingly. In the first approach the expected defect level is defined by collecting actual data about the insertion and detection rates in each phase of the software development project. This defect profile is scaled to fit the size of the project to which it will be applied. Additionally the profile can be adjusted to any change that may have an impact on the performance, such as an improved project goal to reach a specified quality target if previous performance did not yield the required level of quality.

In the second approach the expected number of defects for each phase is generated analytically by following a Rayleigh dispersion curve, which is roughly proportional to project staffing (Card 2003). An analytical model can be used when the organization does not have complete life-cycle defect data, or as an initial solution for new projects without prior historical data.

Usually the comparison of the actual data to the expected data is done at major phase transitions (milestones), but it is recommended to have checkpoints in shorter intervals if the project life-cycle is long (Card 2003). The differences between the expected and the actual figures are not always related to quality problems, but can be a result of, for example, a flawed initial plan, change in the process performance, or incomplete test coverage.

If the measurements are generated for project or company management, the metrics need to be properly validated, sufficiently understood and tightly linked with the attributes they are intended to measure (Kaner and Bond 2004). Otherwise there can be measurement distortions and dysfunctionality in the measurement utilization. With proper models, project organization can support decision making regarding the trade-offs between the four software control variables (Beck 2000): cost, schedule, scope and quality. Defect profile presents

quality performance in an understandable form, and consequences of the decisions (for example, “reducing testing effort to accelerate progress”) can be predicted more easily (Card 2003).

8.5 Quality effect of agile methods

There are studies which indicate that after adopting agile methods, software quality has improved. For example, at Yahoo! (Benefield 2008), feedback on agile transformation was collected from the teams, and 54% of the respondents felt that agile practices improved the overall quality and “rightness” of what the team produced. At Cisco (Lifshitz et al. 2008), members of the 10-person pilot team adopting agile practices felt that product quality was higher, and they were consistently delivering at the end of each iteration. Lee and Yong (2010) reported in their study that the value for the organization in implementing Scrum is higher customer satisfaction in addition to improved quality.

From the agile practices, pair programming has been widely studied, and there are results that indicate that it helps in creating higher-quality code (e.g. Begel and Nagappan 2008, Bipp et al. 2008, Canfora et al. 2007) that contains fewer bugs (Begel and Nagappan 2008). Additionally, pair programming seems to improve knowledge transfer and understanding of the code (McDowell et al. 2002, Begel and Nagappan 2008). Pair programming can be a way to improve teamwork and communication (DeClue 2003) and serve as an informal review process (Goeschl et al. 2010).

Test-driven Development (TDD) is reported to improve both external and internal quality (Sanchzez et al. 2007, Huang and Holcombe 2009). At Microsoft and IBM (Nagappan et al 2008) the pre-release defect density of four products decreased between 40% and 90% compared to similar projects that did not use the TDD practice. At Primavera (Schatz and Abdelshafi 2005), after implementing TDD, the defect count dropped to less than 10 per team, which represents over a 75% improvement in defect rates compared to the previous release. Further results on a lower defect rate with no productivity decrease (Maximilien and Williams 2003), and a decrease in fault-slip-through (Damm and Lundberg 2006) are also available to support the positive quality effect of TDD.

In general, XP practices bring quality improvement but can also increase productivity (Layman et al. 2004, Ilieva et al. 2004). For example, refactoring is targeted to reduce the code complexity, and test automation helps to locate the problems early. There are results that indicate improvements in code quality and productivity (Moser et al. 2008). There are also contradicting results, for example Abrahamsson (2003) did not find reduced defect density after using XP.

At Primavera (Schatz and Abdelshafi 2005), basic agile practices were adopted, including cross-functional, self- organized teams and timeboxed iterations, and the organization experienced a 30% increase in quality as measured by the number of customer-reported

defects in the first nine months following the release. Similarly, a set of agile practices was selected for a pilot at Ericsson (Auvinen et al. 2005) instead of trying to implement a completely agile process. These practices included pair programming, collective code ownership, the planning game and the on-site customer. For assessing the impact of the pilot on code quality, the defect count from the pilot was compared to the defect count from a previous delivery. The analysis showed a 5.5% decrease in defects.

Although the agile practices are an important topic in software engineering, there are only few longitudinal (e.g. Li et al 2011, Goeschl et al. 2010) or large-scale company-wide (Laanti et al. 2011, Petersen and Wohlin 2010) industrial studies that have investigated the impact of agile transformation on software quality in terms of defects and quality assurance. Li et al. (2010) did not find any significant reduction of defect densities or changes of defect profiles after Scrum was used in their longitudinal (three-year) case study of transition from a plan-driven process to Scrum. The results showed, however, that the iterative way of working resulted in constant system and acceptance testing with related defect fixing, which made the development process more efficient by improving the control of software quality and release date as the risk for surprises was reduced. Goeschl et al. (2010) concluded in their two-year study that, based on defect density analysis, utilization of pair programming, test automation, continuous integration and retrospectives can deliver excellent software quality for small to medium sized projects.

In a comparative study on the effect of moving from plan-driven to incremental software development by Petersen and Wohlin (2010), the agile practice implementation included e.g. product backlogs, dependency analysis (called Anatomy Plan), small teams and short timelines. Results show that fewer requirements were discarded, and the number of change requests per requirement decreased, thus the requirements became more stable. Also, the fault-slip-through was improved with enhanced unit and component testing which, after some time, had a positive impact on the maintenance cost. In a large-scale company-wide study by Laanti et al. (2011) at Nokia, most respondents agreed on all accounts with the generally claimed benefits of agile methods, including increased quality and transparency, and earlier detection of defects.

Part III Preparing for the Agile Transformation

This part will present the definition process of the defect metrics, which were later on used for evaluating the impact of agile on the defect management, and a list of possible challenges when transforming the defect management from the traditional software development to the agile software development.

9 EXPLORING QUALITY METRICS TO SUPPORT DEFECT MANAGEMENT PROCESS IN A MULTI-SITE ORGANIZATION

Originally published as

Korhonen K. and Salo O. (2008): Exploring Quality Metrics to Support Defect Management in Multi Site Organization – a Case Study. Proc. 19th International Symposium on Software Reliability Engineering (ISSRE 2008), USA. IEEE Computer Society, pp. 213 – 218.

9.1 Introduction

Defects are a significant factor in software quality. In large-scale software systems the number of defects can be very high, and therefore become difficult to manage. An extra challenge is created by distributed development, when communication and negotiation between different cultures and time zones can cause difficulties (Sandusky and Gasser 2005). On an individual level, it might be easy to loose the focus on what is important for the whole program when development is distributed, hundreds of people are working on multiple sites and projects, and the same problems are not equally visible or considered equally important in every part of the organization. In such a case it is important to have clear goals for the process guiding the work, and use measurements to improve the process and the product (Fenton and Pfleeger 1998).

Industrial studies on multi-site software development organizations report problems caused by co-ordination difficulties (Herbsleb and Grinter 1999) and delays in cross-site work compared to same-site work (Herbsleb et al. 2000). One major problem area in industrial defect management is a lack of commonly agreed defect management methods (Jäntti 2006).

Proposed solutions include improving tool usage (Grinter 1995), adopting better negotiation practices (Sandusky and Gasser 2005), using early quality prediction (Khoshgoftaar et al. 1996) and implementing an organization-wide defect management process (Jäntti 2006). Fenton and Pfleeger (1998) and Basili (1980) further state that the metrics used for defect management should be appropriate for the task. There are industry examples of enhancements in defect management process that improve quality (Mays et al. 1990, Daskalantonakis 1992, Khoshgoftaar et al. 1996). However, further research is needed into field experiences of using quality metrics in supporting a defect management process in a multi-site organization.

The purpose of this chapter is to empirically study the effect of using quality metrics to improve defect management in an organization consisting of several hundred people working in multiple locations. The quality metrics were derived from business goals and they were used in four multi-site programs. The results were evaluated on the basis of the data provided by the selected metrics, and compared to another program that was not using these metrics. The results show an improvement in defect closing speed, a reduction in the number of open defects, and earlier reporting of defects.

The chapter is organized as follows: Section 9.2 provides background information. In Section 9.3, the research design, including methods and context, is defined. Section 9.4 describes the implementation of the measurement project, and in Section 9.5 the empirical results are presented. The chapter is concluded with final remarks.

9.2 Background

9.2.1 Quality metrics for software development

Defining quality criteria and metrics to measure defect-related data can provide support for decision making (Basili 1992). Team member participation is considered important in achieving quality results, and management should direct the organization and individuals towards common goals (Soin 1992). The quality criteria should be motivating and realistic so that people can commit to them, and formulated in a way that everyone can understand them in the same way. It has been suggested that there should be 5-7 critical metrics, and out of those, 2-3 would be related to defects (QAI 1995). There are examples in the literature of defect-related quality criteria (Fenton and Pfleeger 1998, QAI 1995, Basili 1992), but the studies do not include empirical examples of the results.

9.2.2 GQM method

Several mechanisms for defining measurable goals have been proposed in the literature, such as OFD (Kogure and Akao 1983), GQM (Basili 1992, Solingen and Berghout 1999) and

SQM (Boehm et al. 1976). In this study, we have adopted the GQM method, since it can be applied to any process or product. In the SQM approach, the object of study is the final product, and QFD focuses on user requirements.

Solingen and Berghout (1999) define the GQM (Goal Question Metric) method as consisting of four main phases which form the timeline of a measurement project: 1) planning phase, 2) definition phase, 3) data collection phase, and 4) interpretation phase. In the planning phase, the stakeholders of the measurement program are defined. This includes the measurement team itself as well as the project teams participating in data collection. This phase should ensure the commitment of all stakeholders. In the definition phase, the underlying goals for measurement are defined, and the selection of metrics is carried out by defining appropriate questions and metrics that will provide the appropriate answers. The definition phase can be divided into three sub-phases, as suggested by Basili (1992): a) setting the goals, b) refining them into quantifiable questions and c) deriving the required metrics to answer the questions. The third phase, data collection, is for defining and providing tools and procedures for data collection and storage according to the plans. In addition, the collected measurement data is processed into the metrics data. In the last phase, interpretation, all the metrics data should be interpreted correctly, with consideration to the specific context of performed measurements.

9.3 Research method

This chapter reports a case study (Yin 1994) in which four consecutive multi-site programs were studied over a period of three years. The results were compared to a program run prior to these four. The context of the programs was the same: the development involved several hundred people working in three different countries, the goal of the programs was to develop the next release of the same telecommunications software product, and the process model was traditional waterfall.

Action research (Stringer 1999) was applied as a research method and the field study was carried out as part of a program improvement project. Through problem diagnosis it was identified that the number of open defects was too high to manage, and the quality of the software measured against defects was not on the expected level. One of the factors behind these findings was distributed development which produced extra challenges in communication and delays in deliverables. The action intervention decision was to select quality metrics that would motivate everyone in the program organization towards efficient defect management. The author was part of the program team and was therefore able to observe the situation when the selected metrics were taken into use.

9.4 Implementation of the measurement project

The planning of the measurement project was done by the program manager, the quality manager, the author as the program defect manager and project representatives from the Program Organization. The idea was to define quality metrics for one program at a time, and on the basis of the results, fine-tune the criteria for the next program.

9.4.1 Goals and questions

The first goal was to reduce the number of open defects. The questions identified for this goal were: 1) What is the number of defects at certain milestones? 2) Is the number of the defects reduced in every project? 3) What is the defect target we are aiming at? It was agreed that the metrics should also motivate people to report all the defects and provide a basis for progress estimations.

The second goal was to push the defect detection into earlier phases of the program to get more time for fixing and testing the corrections, as suggested also by Boehm and Basili (2001) and Mays et al. (1990). The question derived from this goal was: 1) How many defects were found during different phases of the program? In order to achieve this goal, we should motivate people to see that it is important to find and report defects as early as possible.

The third goal was to speed up the verification of corrections and to be able to determine if the verification activity is a bottleneck for the program. Questions arising from this goal were: 1) What are the verification times in every team? 2) Are they within acceptable time limits? 3) What are the areas requiring immediate action?

The fourth goal was to define the quality metrics in a way that would motivate everyone to co-operate in the fixing of defects regardless of the project or site. Requirement for the metrics was to define a challenging quality metric target level and to give a message that high quality is desirable, but also to keep the target level realistic. One big challenge was how to define criteria so that it would encourage co-operation, and even require it in order to reach good quality results on project level.

9.4.2 Achieving goals with metrics

To give answers to the questions and to fulfill the requirements derived from the goals, we selected three defect-related quality metrics to be implemented. First, *Number of open defects* counted defects for which the correction had not yet been verified. The second metric, *Defect finding profile*, drew a project-specific figure of how many defects were found during different phases of the software development project. The last metric, *Defect lifetime*, measured the time between the points when the defect was reported and when it was marked as corrected and verified in the defect reporting tool. Each metric was an indicator of the

program quality as such, but when the target level was set for each metric, and these three were implemented together as part of the defect management process, they were expected to guide the work into the direction that was set in the beginning.

The selected quality metrics are described in more detail in the following subsections.

9.4.3 Number of open defects

The number of open defects was calculated at two program milestones: first when the system component testing was completed and the system level testing was about to start, second at the end of the program when the mass delivery started. An additional checkpoint was defined for the point when the system testing was completed and the system was ready for pilot deliveries. At this point the number of open defects should be the same as the final target at the end of the program.

Target levels for each defect class were defined separately. There should not be any critical defects open at any milestone; but some major and minor defects were acceptable, depending on the functionality area they were related to. The target level was first set for the whole program at the last milestone to indicate the quality level of the software that would be delivered to customers. After this, the number of defects in each severity class was distributed to projects as individual project targets. The initial quota for each project can be calculated, for example, as a percentage of project work effort from the whole program workload. This value can be fine-tuned according to history data, or other relevant information.

Ideally, no program should have open defects at the first milestone. Often in real business life, this is not possible as schedules are tight and there is not enough time to fix all the late defects reported during component interoperability testing. Also, it might not be possible to include all corrections close to milestones, due to the high risk of breaking something and gaining only little if the correction works.

One specific issue that was agreed within the program was what defects would be counted as open defects. A general rule is that a defect is open until the correction has been verified. However, there are possible exceptions to this rule; for example, how existing defects, found during the previous release, are accounted for in the current program: are they still open defects in this program? How about defects that are agreed not to be fixed in the product? Are they still open defects or features? Program-wide guidelines were needed to deal with special cases.

9.4.4 Defect finding profile

The profile was calculated at the end of the program once all data was available. In the context of this study, a case-specific goal based on data analysis from previous programs was set as follows: at least 80% of the defects should be found during component testing and less than 20 % should be discovered during system-level testing.

Even though the final result can be measured only when all the testing has been done, the data trend line can be compared to previous programs in the same phase to find out differences indicating possible problems.

The real value of this metric for multi-site development was that it steered the project towards reporting all defects found in component testing into the reporting tool, as every reported defect increased the chances of getting the ratio to 80/20. This is helpful, for example, in a case where a subcontractor might not want to report all their defects into public tools because of their number-of-open-defects quality target.

9.4.5 Defect lifetime

This metric indicated how long the defect had been open before it was fixed and verified. According to Fenton (Fenton and Pfleeger 1998), the longer a defect stays open in the system, the more irrelevant it becomes. Projects followed up closely the defect closing speed in each severity class from the defect lifetime data. Long defect lifetime could indicate a laborious fix to the defect or delays in verification. Additionally, by comparing the number of open defects and the current closing speed it was possible to give an estimate on how long it would take to close all open defects in the project.

Target lifetime level was defined based on history data from previous programs, and same target level was applied to every project. Severity classes have different urgency and therefore each class should have a separate target time limit. Fixes for the critical defects are expected to be available faster than to major or minor class defects. In practice, there were easy and laborious fixes from all of which the metric was calculated as an average time in each class. Agreed target values were not to be exceeded.

Defect life time could also be calculated from new to corrected, according to reporting tool status, if correction and verification were done in a different projects. But when the lifetime was calculated for all defects from new to verified, it seemed to provide better results in a multi-site environment. In this case, the project fixing the defect had an additional interest to request the verification from the project team who found the defect in the first place, as a delay in verification would have increased the lifetime of the defect on their responsibility. This increased the co-operation between teams and thus, sped up the verification time

9.4.6 Mapping metrics to goals

As shown in Table 3, all metrics supported the main goal Goal 1: *Reduce the total number of open defects*, from different aspects. The *total number of defects* metric gave a clear number for projects to target their efforts to, the *defect finding profile* metric provided motivation to report all defects to the reporting tool as soon as they were found, which again gave more time to analyze and fix the defects. The *defect lifetime* metric sped up the process of handling

the defects, since fixing and testing the defects was done as soon as possible to meet the target time limits.

Table 3. Summary of selected metrics versus goals.

Metric\Goal	Goal 1	Goal 2	Goal 3	Goal 4
Total number of defects	x		x	x
Defect finding profile	x	x		
Defect life time	x		x	x

Defect finding profile gave a good result to the project if at least 80 percent of the defects were found during early phases of the program, and this definitely pushed the *defect detection into earlier phases* of the program as set in Goal 2. The total number of open defects supported Goal 3: *speed up the verification*, as a defect was counted as open for the project until it was verified. Also the metric Defect lifetime sped up the verification and therefore supported Goal 3. Goal 4: *motivate to co-operate over sites* was covered by the metrics Total number of defects and Defect life time. The total number of defects metric encouraged co-operation when the verification or even part of the correction was done by other projects. Defect lifetime motivated teams to work together in order to get the verification between different teams as fast as possible.

9.4.7 Metrics as part of the defect management process

The selected metrics were implemented as part of the defect management process of the program. Statistics for each metric were calculated every other day, and to get better visibility throughout the program, the status reports were distributed by email to the Program Organization. Metrics were collected and distributed regularly already from the beginning of the component testing phase. In each project, data was analyzed and actions were taken immediately when problems were noticed. Program management followed the progress of the defect management and contacted project managers and team members directly if there were deviations from the targets. The metrics helped in milestone decision making as the milestone could not be approved for the project and the program if the target levels were not met.

Even though the data was mainly used for locating problems, they also communicated positive development in the projects to the whole Organization, e.g. if a project improved their progress compared to the previous week, or was able to perform better than the target level. Also, based on the data collected from previous programs, it was possible to analyze and estimate how the fault status would and should develop in the ongoing program in order to reach the target level.

9.5 Empirical results

Data for the metrics was collected with a defect reporting tool which was used by every member in all programs reported in this study. Using pre-defined data was mandatory in reporting a new defect, thus the data was consistent and comparable between the programs.

Figure 4 shows a comparison of open defects in programs 1-5. There were three main findings. First, the number of open defects was constantly lower in programs 2-5, where the selected metrics were in use. Program 1 was the base version, and later programs were implemented on top of that, which partly explains the sharp drop of defects from program 1 to 2. But also, in program 1 there was a challenge with slow closing speed, and thus, open defects were accumulating. The second finding was that the trend line became more predictable in programs 2-5, and there was no high peak in defects during system testing, but the trend was consistent. The high peak in program 1 during the system testing phase was mainly caused by installation problems as the installation was not tested until that point. Both of these problems were taken into account in programs 2-5 by following up the defect closing speed and testing the installation already during the component testing phase. The third finding was that the same level of open defects was achieved in programs 2, 3 and 5 before the system testing phase, regardless of their initial defect numbers. The trend line in program 4 is slightly different due to the fact it had one additional system testing round 2 weeks before the official system testing started and this revealed new defects.

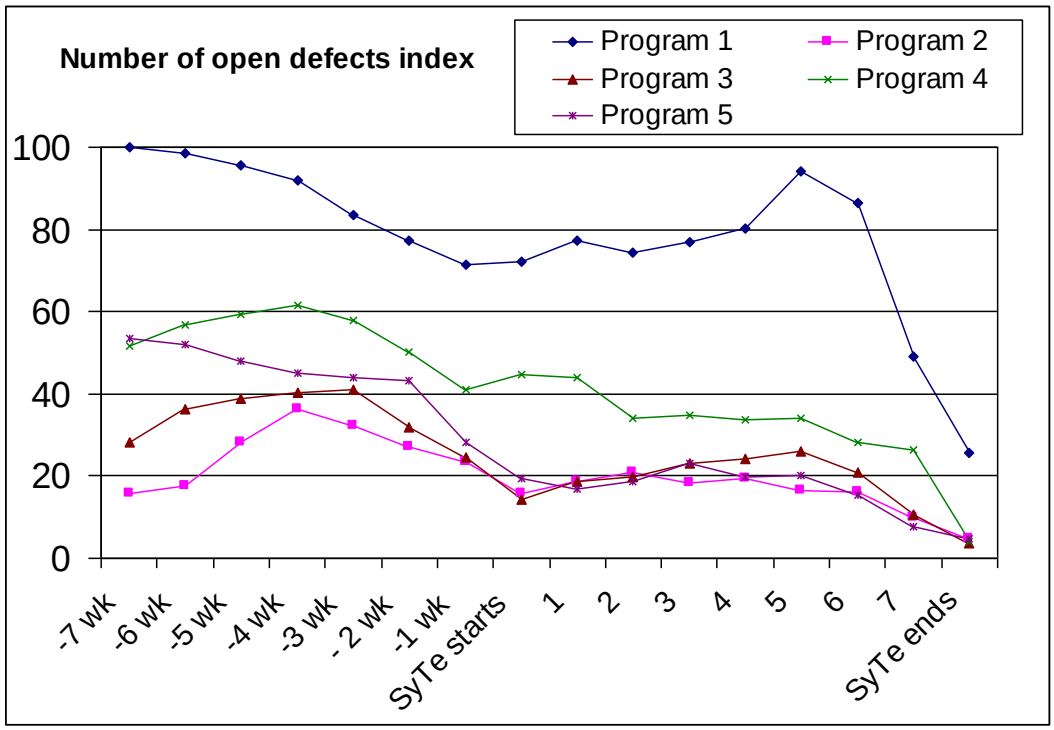

Figure 4. Open defects trend line.

Figure 5 shows the defect finding profile of programs 1-5. The results show that the goal of pushing the defect detection to the component testing phase was achieved in programs 2-5. Starting the installation testing already during the component testing phase helped reach this goal, and the defects related to the installation were counted into component testing figures in all programs to get comparable results. However, for individual projects within the programs, the installation defects were left out from calculations for the milestones as they were considered to be too laborious for the projects to test on component level.

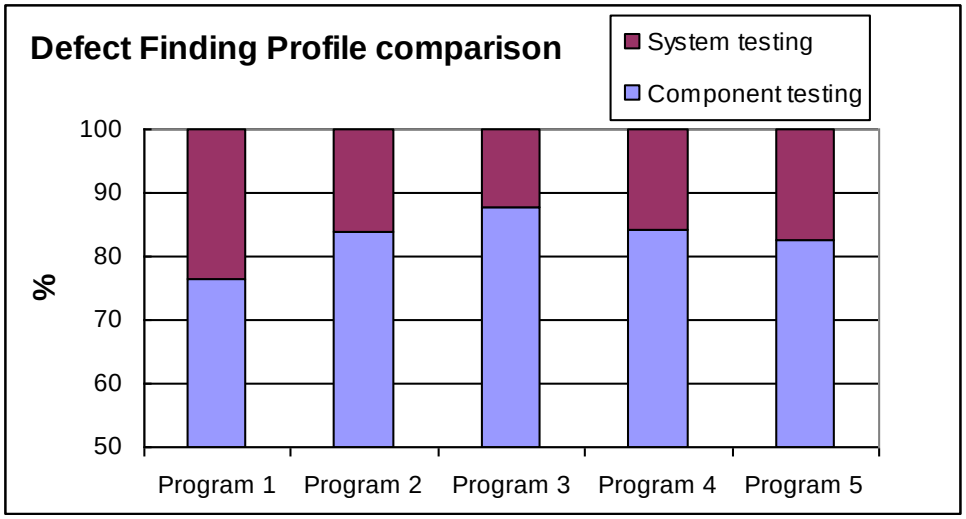

Figure 5. Defect finding profile

Defect lifetime was measured separately for each defect class by calculating how many of the defects were fixed and verified within the target time limit. Time was calculated in days including both correction time and verification time. Figure 6 shows the percentage of the corrections of the major class defects completed before this time limit. The results show some improvement in programs 2-5 compared to program 1. Similar results were achieved with the minor and critical class defects as well.

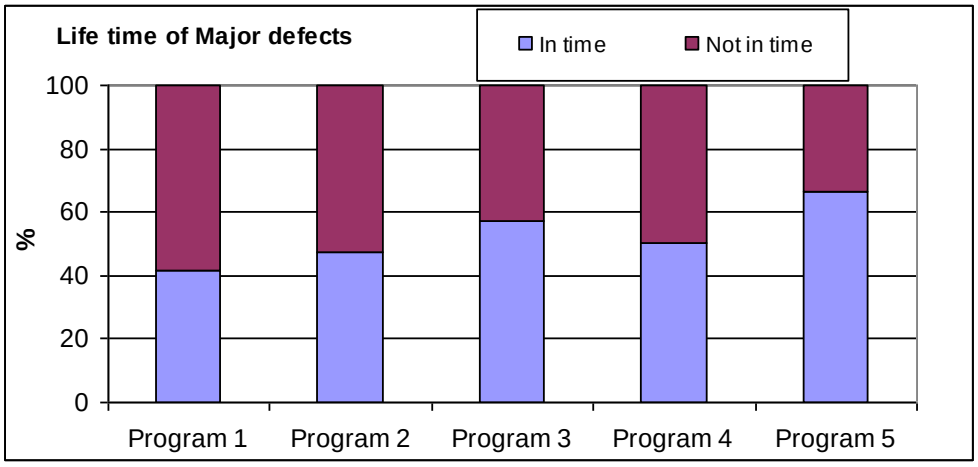

Figure 6. Defect lifetime analysis for major defects

9.6 Conclusions

The main contribution of the study in this chapter lies in providing empirical evidence that realistic quality targets derived from the organization goals and followed up with metrics can guide the defect management work and help to get quality results in large multi-site software development programs. The measurement data supports the conclusion that the selected metrics indeed had an effect on the results. Having the criteria and following the progress regularly also changed the defect management process within the Organization. Previously the number of open defects had been followed up carefully, but now also the defect finding profile and the defect closing time were added to regular follow-up. The follow-up was done on the project level and by the different sites. This focused the effort of the projects in all sites as the results were all the time visible to every team member, and if the figures were not according to the targets, the project manager was asked to explain the situation to the program management.

Experiences from this measurement project showed that in order to support the defect management process with the quality criteria, the target levels for the metrics should be realistic, metrics need to be followed up on a regular basis and the progress should be made visible to everyone. It is also required that the management is committed to getting the quality results and the milestones are not approved unless the quality criterion is met.

Defining the quality criteria to support the defect management process can help only to a certain extent – a lot also depends on the communication between the projects and the teams on different sites and on how the software development program is organized.

To confirm the validity of the results, additional research methods, e.g. a survey, could have been used, and the research on this topic can proceed into that direction. In future studies it could be explored what other combinations of quality metrics could prove effective in supporting multi-site work, what kind of differences in the organizations can have an impact on the results, and whether applying these metrics to smaller software development programs would provide similar results.

10 MIGRATING DEFECT MANAGEMENT FROM WATERFALL TO AGILE SOFTWARE DEVELOPMENT IN A LARGE-SCALE MULTI-SITE ORGANIZATION

Originally published as

Korhonen, K.(2009): Migrating Defect Management from Waterfall to Agile Software Development in a Large-Scale Multi-Site Organization: a Case Study. Proc. 10th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP 2010), Italy. LNBIP 31, Springer, pp. 73-82.

Part IV Evaluating the Effect of Agile Transformation

This part will present the evaluation of agile transformation impact on defect data and defect management practices. The engagement to the agile transformation process is discussed, and the agile practice adoption between different teams is presented. This part is concluded with evaluation of the agile transformation success.

11 EXPLORING DEFECT DATA, QUALITY AND ENGAGEMENT DURING AGILE TRANSFORMATION AT A LARGE MULTISITE ORGANIZATION

Originally published as

Korhonen, K. (2010): Exploring Defect Data, Quality and Engagement during Agile Transformation at a Large Multisite Organization. Proc. 11th International Conference on Agile Processes and eXtreme Programming in Software Engineering XP 2010, Norway. LNBIP 48, Springer, pp. 88-102.

Published in dissertation with kind permission of Springer Science and Business Media.

12 EVALUATING THE IMPACT OF AGILE ADOPTION ON THE SOFTWARE DEFECT MANAGEMENT PRACTICES

Originally published as

Korhonen, K.(2011): Evaluating the Impact of Agile adoption on the Software Defect Management Practices. Software Quality Professional volume 14, issue 1, December 2011.

Published in shorter format as

Korhonen, K.(2010): Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting Practices. Proc. 7th International Conference on the Quality of Information and Communications Technology (QUATIC 2010) Portugal, IEEE Computer Society, pp. 35-43.

12.1 Introduction

Recent studies propose that agile practices, such as pair programming and continuous integration (Lindvall et al. 2004), help to improve code quality and reduce the number of defects (Auvinen et al. 2005). This has also encouraged large software organizations to begin agile transformation, although large (Larsson 2003) and distributed organizations (Misra et al. 2006) are one of the challenges in agile adoption. The impact can be seen, for example, in difficulties managing the defects (Korhonen 2009). Due to these challenges, it is essential that an organization evaluates the actual effects of agile transformation to react accordingly during its agile adoption.

Previous studies have used various methods to show that quality has improved with agile adoption. These methods include analyses of the number of defects (Schatz and Abdelshafi 2005), formal code reviews (Auvinen et al. 2005), and surveys exploring the perceptions of quality that people in the organization have (Laanti, Salo, and Abrahamsson 2011). However, the changes in defect reporting practices due to agile adoption, and the impact of those changes on defect data, are not further analyzed in the studies. Understanding the impact of agile practices in defect reporting becomes important especially when doing defect-based code quality analysis.

In this study, the goal was to explore the agile effect on software defect management in a large, distributed software Organization. The effect was studied from two viewpoints: 1) changes in the defect data; and 2) changes in the defect reporting practices. The main source

of data was the defect data records. Additional information was gathered by interviewing the key members of the project team before the transformation started, and with two surveys during the first year of agile.

The analysis showed that, though there were challenges in migrating the defect management to agile, the defect data were behaving as expected; for example, defects were reported on regular speed over time, and the number of defects was reduced. Adopting agile practices changed the defect reporting practices, which can to a certain extent explain the change in the defect data. Therefore, one might claim that the change in reporting practices toward a desired way of working may imply a positive impact on the quality as well.

Based on the results, it is proposed that organizations transforming to agile should study the defect data and changes in their defect reporting practices regularly during the agile transformation to find evidence of the effect of agile, and to verify that the changes are consistent with their expectations. The changes in defect data metrics can show the trend as to whether the impact is what was expected, and the analysis may clarify the background of the changes, revealing, for example, improvements needed in the defect management practices.

12.2 Background

One agile viewpoint proposes that defects are seen as waste (Poppendieck and Poppendieck 2007). Defects should be fixed as soon as they are found and, therefore, reporting the defects with a tool should not even be required. According to this view, an immediate result of agile adoption would be a reduction in the number of defects. Such results have indeed been reported in previous studies (Lindvall et al. 2004). Further, as a consequence of new content being developed and defects found on regular speed in every sprint (Crispin and Gregory 2009), it can be expected that the defect closing time should not be longer than one sprint in agile projects, and there should not be any major peaks in defect reporting over time.

Agile practices, such as automated tests and test-driven development (TDD), are reported to help improve product quality and reduce the number of faults as well (Nagappan et al. 2008). A set of agile practices (pair programming, collective code ownership, the planning game, and the on-site customer) was selected for a pilot at Ericsson (Auvinen et al. 2005) instead of trying to implement a completely agile process. The analysis showed a 5.5 percent decrease in defects reported from a pilot. At Primavera (Schatz and Abdelshafi 2005), basic agile practices, including cross-functional self-managed teams and time-boxed iterations, were applied and the organization experienced a 30 percent increase in quality as measured by the number of customer-reported defects in the first nine months following the release. After implementing TDD, the defect count improved even further. However, these positive results also show that although the number of faults has decreased, faults still exist and the remaining ones need to be handled systematically.

Managing defects online during the same sprint may not be an issue in the case of a colocated software development scrum team (Schwaber and Beedle 2002). In a distributed organization, however, the teams are working in different physical locations, so there may be communication problems as well as cultural differences. Additionally, if the software system is old, the legacy code base is likely to have an unknown number of defects. Therefore, a zero fault tolerance at the end of the sprint may not even be possible, as the cost of the correction with respect to the business value of the correction is not on a feasible level. These aspects have made visible a need for more formal communication between teams than agile method proponents originally suggested (Lindvall et al. 2004), and one proposed solution is to use the defect tracking tool as a defect backlog (Korhonen 2009).

Success stories of adopting agile methodologies highlight the benefits of the transition to agile development, but also give examples of possible challenges. Common pitfalls in migrating to agile and effective approaches for facilitating the change are described in several studies (for example, Cohn and Ford 2003). Among other issues, distributed development has proved to be challenging, and it is proposed that it should not be used during the first two or three months after initiating an agile process (Misra et al. 2006). If the organization is already distributed, this requirement is not easy to follow, but there are also encouraging reports of agile transformation done in a large, distributed organization (Korhonen 2009). It just requires special attention in this respect; for example, agile transformation can be started within small, co-located teams first (Auvinen et al. 2005), and then advanced to other locations in the organization.

12.3 Research setup

12.3.1 Research context

The Organization in the study was large, with more than 150 experts working in the studied, globally distributed projects. Most of the people had several years of experience in traditional software development, and some even had previous experience with agile software development gained from working in other parts of the company. The software development projects produced new versions of an existing product in the telecommunications domain. During the one-year study period, agile transformation was gradually started in the studied Organization by first implementing the basic agile practices.

12.3.2 Research questions and data collection

The primary goal of this study was to provide answers to four questions:

- What changes were visible in the defect data during the first year of agile transformation?

- What kinds of changes could be seen in the defect reporting practices?
- Were these changes consistent with the expectations based on the agile literature review?
- What kinds of challenges were there in defect management?

Three sources of information were used: defect data records, interviews before the agile transformation, and two surveys during the agile transformation. The defect data were collected before and during the first year of the agile transformation from three software development projects (project A, B, and C) once a week from the defect reporting tool. The three projects were similar in size based on source lines of code and complexity. At the end of each project, the defect data were analyzed first by creating and comparing some basic metrics. A second analysis was done on the defects in different severity classes. Project A was the last project in waterfall mode before the agile transformation started. Project B was the first one to take agile practices into use, and the related data were recorded six months after starting the agile transformation. The data from an agile project C were recorded 12 months after starting the agile transformation.

Before the agile transformation started, there was an initial interview with project team members to identify potential challenges in defect management during the agile transformation (Korhonen 2009). To get a wider understanding of the possible problem areas, the interview was conducted additionally in two other organizations, which were more mature in the agile practice adoption.

While the agile transformation proceeded, the management wanted to collect feedback on the progress from the project personnel with a survey. Additional questions related to defects, and the challenges identified during the interviews, were combined into the feedback survey, and these questions were analyzed for this study. Afterward, the survey results were combined and discussed in feedback sessions arranged by the author with the project members and management. In these sessions, the results were shared and discussion was initiated on both the reasons behind the results and to agree on an action plan for the findings prioritized by the project members.

12.4 Empirical results

12.4.1 Adopting the agile practices

To get a better understanding of how the agile transformation was proceeding in the Organization and what changes took place, a question on agile practices in use was included in the survey. The responses are summarized in this section.

The practices are divided into two categories to visualize the change more clearly: daily practices (Figure 11) and technical practices (Figure 12). Daily practices, such as time-boxed iterations, product backlog, and scrum, are adopted by the whole Organization, forming the

framework for the daily work. Technical practices, such as continuous integration, refactoring, and TDD, are directly related to code development.

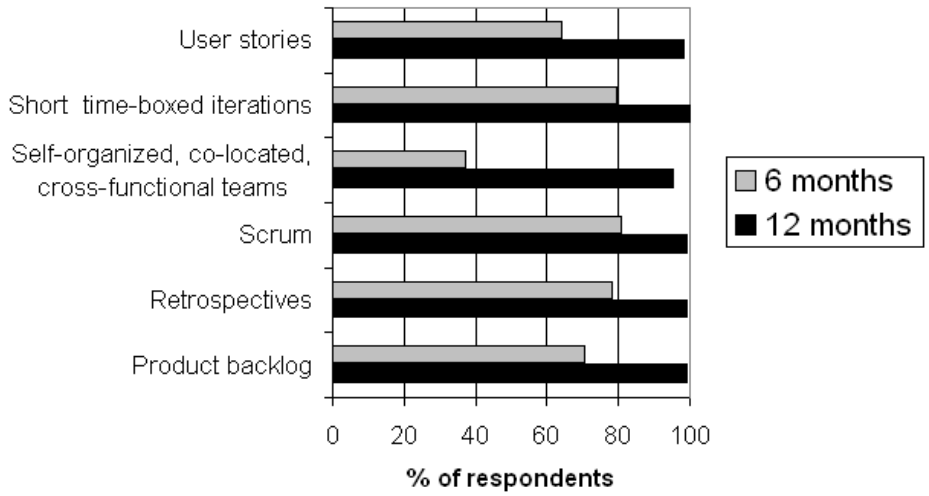

Figure 11. Daily agile practices in use six months and 12 months after starting the agile transformation.

As displayed in Figure 12, during the first six months after starting the agile adoption, the Organization was in a very early stage of agile adoption (Vilkki 2009) – scrum teams, iterations, and retrospectives were reported to be in use, but even these were not used by all of the teams.

During the next six months, the situation had clearly advanced. Time-boxed iterations, retrospectives, scrum teams, product backlog, and user stories were reported to be used by every respondent.

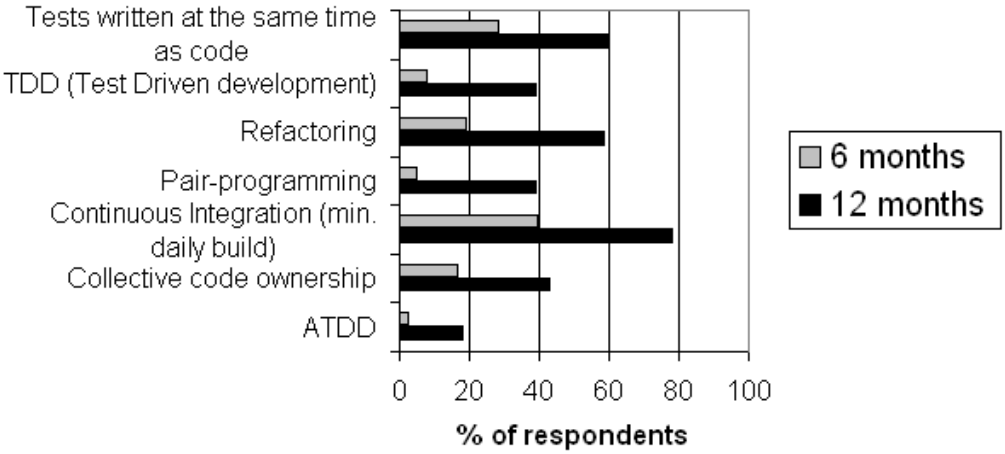

Figure 12. Technical agile practices in use six months and 12 months after starting the agile transformation.

After 12 months of starting the agile transformation, the respondents also utilized more technical practices (Figure 12). For example, one of the key things in an agile way of working, continuous integration, had increased from 40 percent to nearly 80 percent during the latter six months. One hundred percent utilization of the technical practices was not expected, as the participants in this study included people with no direct coding responsibility. Figure 9 describes the utilization of the code development related practices within the developers after 12 months.

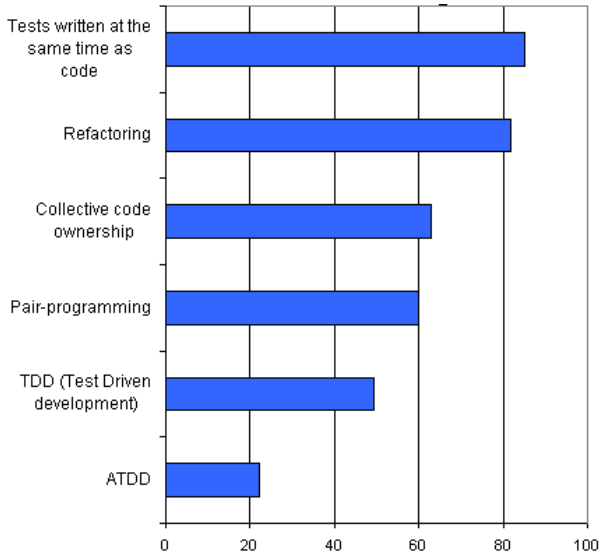

Figure 13.Technical agile practices in use in the teams developing code 12 months after starting the agile transformation.

12.4.2 Challenges in defect management during agile transformation

The analysis of the interview results revealed several possible problem areas in defect management during agile transformation related to process and tools in a multisite organization (Korhonen 2009). Respectively, the following recommendations should be considered (Korhonen 2009): 1) specify faults to be reported; 2) create practices for prioritizing between fault fixing and feature development during a sprint; 3) evaluate the appropriateness of the current fault reporting tool and related practices; and 4) evaluate the multisite development impact on defect management.

Related questions were added to the survey to follow up if these areas were seen problematic in the Studied Organization during the first year of agile transformation. The results are discussed next.

12.4.2.1 Which faults need to be reported?

The confusion on the need to report the defects comes from the conflict between the expectation and reality. The expectation is that, while in agile, one should focus on fixing the defects, and thus the reporting of the defects into the tool would not be required. But often in reality, it might not be possible to fix all of the defects at once, such as those that are the other teams' responsibility. Proposed actions (Korhonen 2009) were to make organization wide formal and written guidelines on how to handle the defects in these special cases and to introduce fault reporting tools as a fault backlog.

According to the survey responses (Figure 14), it was not clear how to handle the defects in agile, especially in the beginning of the transformation. During the first six months of the transformation, less than 50 percent of the respondents thought they knew how to handle the defects in agile. After the first survey, there were additional written guidelines introduced in the Organization on what defects to report--for example, open defects at the end of the sprint or defects to other teams should be reported with the tool. Either this clarification, or people being more experienced in agile, helped to improve the situation, as after 12 months, more than 70 percent answered that they knew how to handle defects in agile. On the other hand, there were still people who were not clear on the practices, and this would require further attention.

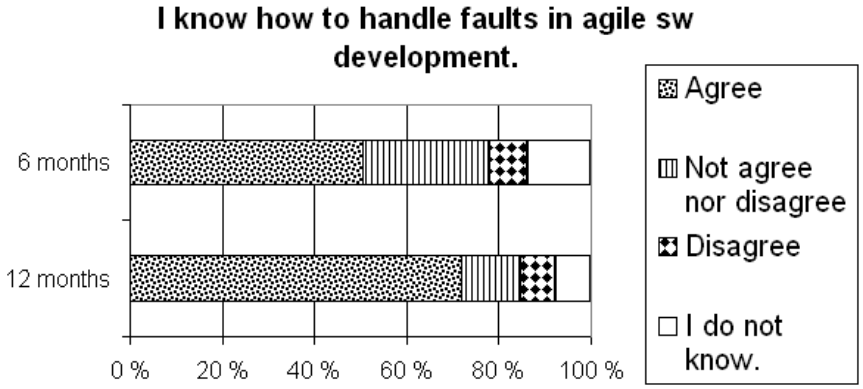

Figure 14. Survey responses to statement "I know how to handle faults in agile software development."

12.4.2.2 Prioritization of fault corrections

Prioritization of the defect correction over new content development may also result in a conflict between expected practice to fix defects as soon as they are found (Poppendieck and Poppendieck 2007) and reality. In reality, it can be difficult to prioritize between the fault corrections and feature development during one sprint due to pressure from the different

stakeholders, as reported in previous studies (Korhonen 2009; Schatz and Abdelshafi 2005). Proposed (Korhonen 2009) ways to overcome the situation would be to: 1) establish a common process for how to prioritize between fault fixing and feature development; 2) allow minor fault fixes to be transferred to the next sprint; and 3) schedule fault fixing sprints. The latter two can be helpful, especially in the early phase of agile adoption, as long as the long-term target is to fix the majority of the defects during the same sprint to avoid the quality debt.

In this Studied Organization, the priority between fault fixing and new content development seemed quite clear right from the beginning. According to more than 60 percent of respondents in both surveys, fixing the faults was seen as more important than developing new content (Figure 15). But there were also people who thought new content development would be more important (10 percent in both surveys). One participant in the feedback session commented that in reality there might be a situation where developing certain new, customer-critical content might have a higher priority than fixing minor class faults in other, less customer-critical areas. Thus, setting the priority higher on fixing the defects than developing new content would also be case sensitive, according to the respondent.

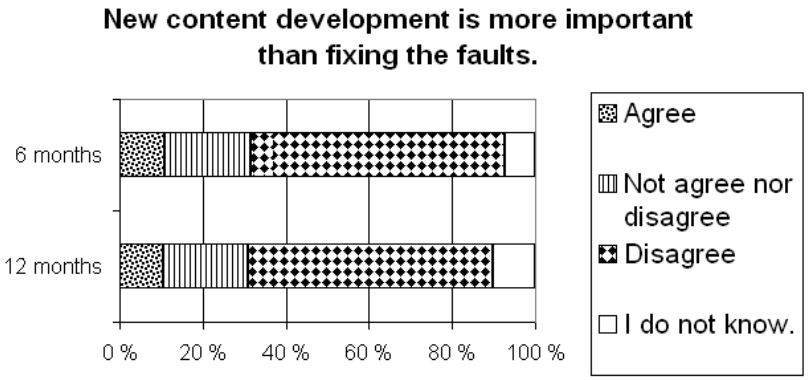

Figure 15. Survey responses to statement "New content development is more important than fixing the faults."

12.4.2.3 Fault reporting tool

As the way of working is changing, the requirements for information inserted in the defect reporting tool for each defect might need to be enhanced. The proposal (Korhonen 2009) was to evaluate the appropriateness of the current tool and tool-related practices before the agile transformation, and based on the findings create common guidelines on how to use the tool in agile, and especially what data need to be inserted for each defect.

The related question in the follow-up survey was whether the respondent thought the fault reporting tool in use was suitable for the agile methods. The feelings were mixed (Figure 16). After six months, 40 percent thought the tool was suitable for agile methods, and 22 percent thought it was not suitable. The clarification that was done on the defect reporting practices after first six months of the transformation might have helped, as 12 months later the situation

had improved a bit. Still, however, 16 percent of the respondents thought the tool was not suitable for agile methods. One comment in the survey responses was that it takes too long to report the defect with the tool, as there were too many mandatory fields to fill in. The number of mandatory fields had not been changed before or during the agile transformation.

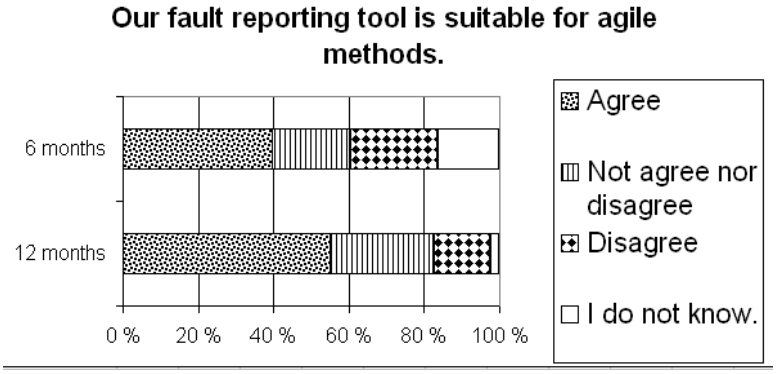

Figure 16. Survey responses to statement "Our fault reporting tool is suitable for agile methods."

12.4.2.4 Distributed development

Dependencies between teams on different sites can cause extra challenges to agile transformation in a large multisite organization, which requires special attention on communication practices. For example, daily scrum meetings would require extra effort to arrange for participants from different locations. Based on the interview results (Korhonen 2009), it was proposed that an organization should become familiar with agile first and only after some experience with agile, introduce the multisite work. On the other hand, if development was already done in multiple locations, the organization could consider evaluating the defect management related communication practices to locate places for improvement.

According to the survey results, the impact of agile transformation on cooperation between the teams in different locations (Figure 17) was mostly seen as positive. During the first six months, 55 percent of the respondents, and 57 percent during the latter six months of the transformation, thought the cooperation, communication, and teamwork of teams in different locations had improved. There were, however, people who believed it had gotten worse; for example, it was difficult to analyze if the bugs a team found were already found by other scrum teams.

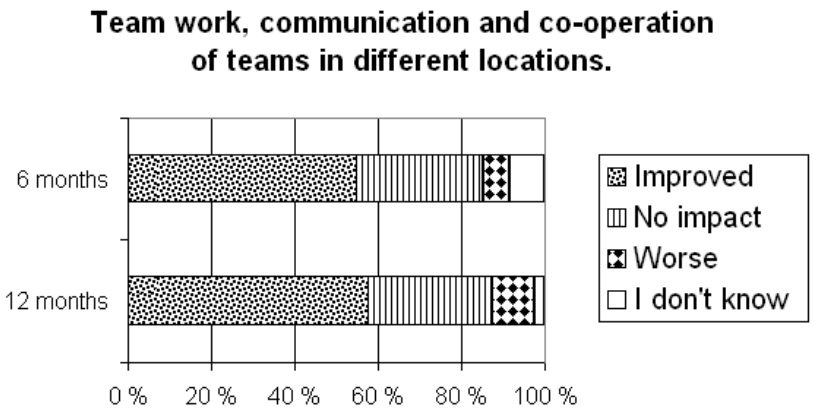

Figure 17. Survey responses to question if the teamwork, communication, and cooperation of teams in different locations have improved.

12.4.3 Analysis on defect data and defect management practices

In this section, first the defect data metrics are presented for: 1) describing the change in the number of open defects; and 2) comparing defect lifetime between the projects. Second, the changes in the defect reporting practices are discussed.

12.4.3.1 Changes in defect data

The collected number of open defects from project B and project C were compared to the maximum number of open defects in project A to get the open defect index for each project (Figure 18). The defect data comparison to source lines of code is not presented in this article due to confidentiality reasons. The lines in Figure 18 show the data for a few sprints before and a few sprints after the milestone (vertical line) when the software was frozen for the final system-level testing.

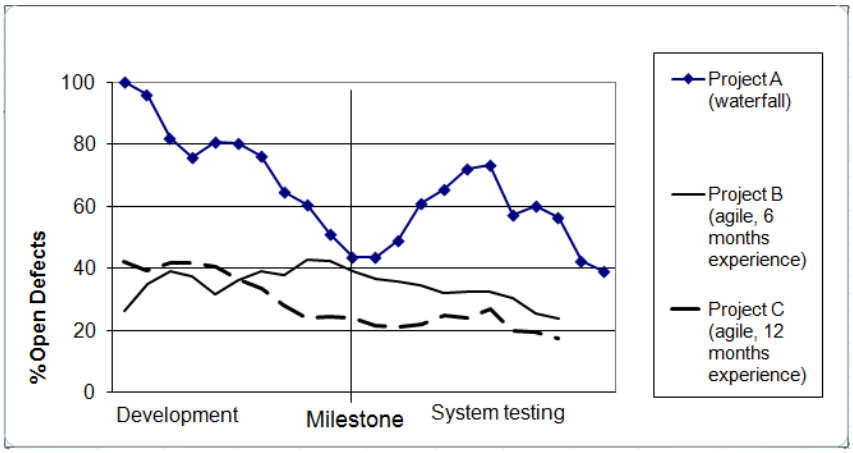

Figure 18. Open faults index. The number of open defects in projects B and C is compared to the maximum number of open defects in project A.

As shown in Figure 18, the total number of defects was lower in projects B and C compared to project A. The defects were also reported on more regular speed over the time in the latter projects. When compared to the nonagile project A, there are no such high peaks in the number of open defects in the agile projects B and C before or after the software freeze milestone.

Defect closing time measures the time in days between reporting the defect and marking the correction as tested in the defect reporting tool. In Figure 19, the defects are grouped according to the closing time into three cumulative groups: defects that were closed: 1) in time less than or equal to the length of one sprint; 2) in time equal to max two sprints; and 3) in time equal to max three sprints.

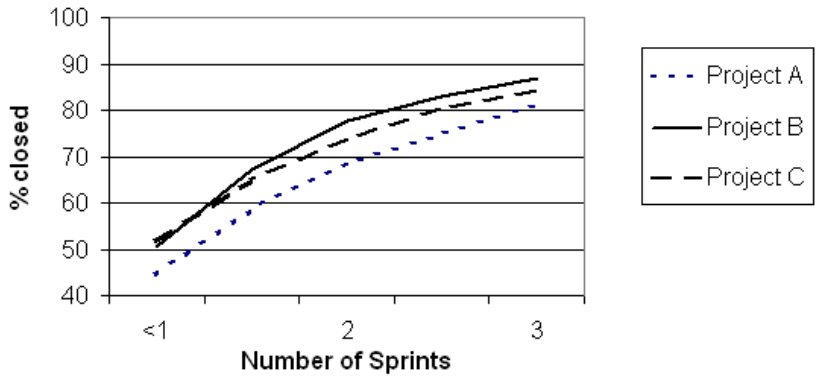

Figure 19. Defect lifetime analysis for projects A, B, and C.

In the agile projects B and C, more than 50 percent of the defects were closed in time equal to one sprint or faster. In the non-agile project A, only 45 percent were closed in that time. On the other hand, in project A it took time equivalent of three sprints to close 80 percent of the defects, whereas the same result was achieved in the agile projects in less than two and a half sprints.

12.4.3.2 Changes in reporting the defects

The defect data were further analyzed from two perspectives: 1) was there any change in reporting the defects between the teams; and 2) was there any change in reporting the defects in different severity classes.

The percentage of defects reported to own team versus defects reported to other teams changed over time, as seen in Table 14. In project B, more defects classified as major, but fewer critical and minor defects were reported to the own team compared to project A. In project C, the number of defects reported to the own team increased in all severity classes.

Table 14. Percentage of defects reported to own team.

Project	Defect severity class		
	Critical	Major	Minor
Project A	2%	17%	39%
Project B	0%	27%	33%
Project C	10%	28%	54%

A comparison between the total numbers of defects in different severity classes did not show any significant difference between the projects in general (Figure 20). In project C, the share of major defects was slightly bigger than in the project A and the project B.

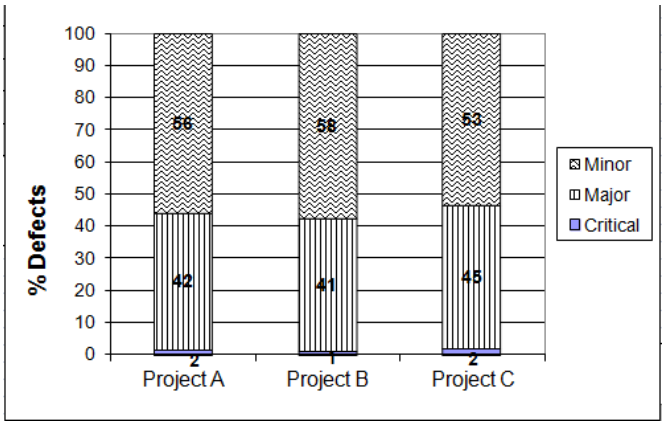

Figure 20. Total share of critical, major and minor defects of projects A, B, and C.

The time when defects were reported was analyzed to see if there were any differences between the severity classes Figure 21 presents the number of minor class defects as a percentage from the highest defect count in a timeline for each project.

In projects B and C, defects in all severity classes were reported on regular speed over time. In project A, most of the critical defects were reported in the earlier phases of development, and more minor defects were reported after the software freeze for system testing, seen as a higher peak in minor defects after the milestone (Figure 21).

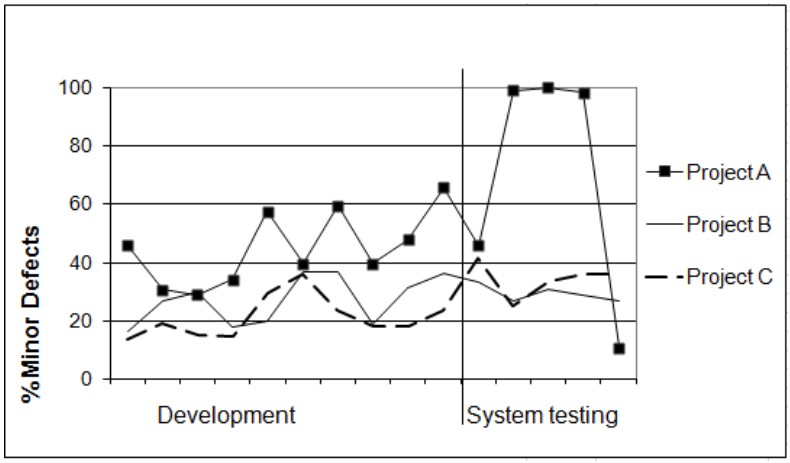

Figure 21. Weekly number of reported minor defects over time in projects A, B, and C.

12.4.4 Feedback from the survey

Responses to two survey questions are presented in this study to support the analysis of the defect data: 1) improvement in early detection of defects; and 2) change in reporting defects with the defect reporting tool.

Improvement in the early detection of defects was reported in both surveys by 57.5 percent of the respondents after six months, and by 65 percent after 12 months (Table 15). Some of the respondents (7.8 percent after six months and 10 percent after 12 months) thought the detection of defects had become slower or more difficult, and the reasons behind these answers were seen as important to be analyzed further. Information was collected with open-ended questions in the survey and from comments during the survey feedback sessions.

Table 15. Survey responses to the question: *Improvement in early detection of defects.*

Agile experience when the survey was done	Early detection of defects			
	Improved	No impact	Gotten worse	I do not know
6 months	57.2%	26%	7.8%	9.1%
12 months	65%	22.5%	10%	2.5%

The most negative feedback came from the people responsible for the verification. The testing procedures in agile were unclear and the length of the sprint created greater pressure to get testing done in a shorter time period than before. Testers thought there were too many test cases to be run manually in this short time, and a proper test automation environment was missing. Additionally, there were problems with continuous integration, the installation took

too long, and by the time it got installed for testing, the software was already “too old.” Even only a few days’ delay was a big thing--during that time, the development teams had developed new code and fixed bugs in the software.

Sometimes testing was not possible because a dependant component of the functionality was missing; thus, some bugs in the team’s “own” code were left undetected until the missing part from the other development team was available and integrated. In large-scale software projects, there are usually dependencies regardless of the software development method. But in this case, agile practice, that is, short time-boxed iterations, brought this problem painfully visible to the testers, and they felt the early detection of defects had gone worse.

In the survey, the respondents were asked whether they report all of the defects they find with the defect reporting tool. After six months, 51.1 percent of the respondents reported all faults they found, while 26.6 percent did not (Table 16). After 12 months, 80 percent of the respondents said they reported all defects, and 17.5 percent said they did not report all the defects with the reporting tool.

Table 16 Survey responses to the question: Reporting faults with the fault reporting tool.

Agile experience	I report all faults I find with the fault reporting tool			
	Agree	Not agree or disagree	Disagree	I do not know
6 months	51.1%	16.7%	26.6%	5.6%
12 months	80%	2.5%	17.5%	0%

Based on the feedback session comments and the responses to the open-ended questions in the survey, the majority of the respondents thought it was unnecessary to report defects that they could fix right away in their team, or that it was faster to get the fix by walking to or calling the developer in another scrum team directly. Some respondents thought reporting minor class defects could be a waste of time, as those did not always get enough priority to get fixed during the ongoing sprint, and thus getting the correction would be delayed.

12.5 Discussion

12.5.1 Defect data

The fact that the defect reporting tool remained the same, and there were no changes in the mandatory information to be inserted for each defect data record in the tool, makes it possible to compare the defect data records for this study before and after the agile transformation.

The empirical results indicate that there had been changes in the defect management after the agile transformation was started in the Organization. The number of defects reported remained on same level in every sprint, which is an expected result based on literature (Poppendieck and Poppendieck 2007). Also, the majority of the project personnel believed that an early detection of defects improved along with the agile transformation.

While the agile development teams work in sprints that have a fixed length and the goal is to provide good quality at the end of each sprint, the expectation is that all of the problems found should be solved during the same sprint. The impact, therefore, should also be visible in the defect closing time, as more and more defects are closed within time equal to one or two sprints. In this study, defects were closed faster in the agile projects than in the traditional project.

One of the challenges in agile is to prioritize between the fault fixing and new feature development (Korhonen 2009). In this study, the priority order was clear, and defect fixing was considered as the first priority by a majority of the respondents. But there were also examples of cases where, for example, the requirement to support customer-specific feature development was seen as more important than fixing the minor class defects that were not related to any customer-critical functionality. Prioritization based on the customer's needs and requirements is one of the agile principles, so it is necessary to take this into account when prioritizing the defect fixing. One the other hand, one needs to be cautious of the impact of this activity, as it may lead to quality issues.

As new content is developed in every sprint (Poppendieck and Poppendieck 2007), there can be a variety of defects found in each severity class. In waterfall, the new content is developed more toward the beginning of the project, and that is also when the most of critical defects are found. The results of this study show a similar trend in the number of open defects.

The agile procedures for testing were not clear to every test engineer, and there were even some responses in the survey that the early defect detection had gotten worse while in agile. Identified problem areas were brought up by the need to do things faster than earlier: dependencies, delays in installation and too many manual test cases preventing the completion of the testing in time, and finding the defects early enough to get the corrections during the same sprint. Indeed, the test engineers were already proposing to add more test automation instead of manual test cases, talking about the importance of continuous integration and how to reduce the dependencies--all of the things that are important from an agile software development point of view (Poppendieck and Poppendieck 2007). By making the issues visible with the short time-boxed iterations, the change forced people to start thinking of new ways, in this case agile practices, to deal with the situation.

12.5.2 Defect reporting

The adoption of agile practices improved continuously during the first 12 months. Changing the way of working caused some confusion on how to handle the defects, and after the first six months, additional effort was put to clarify the guidelines on what faults should be

reported, and how to use the defect tracking tool in agile. This had a positive impact, and awareness of how to handle the defects in agile improved toward the end of the study period.

As agile methods promise code quality improvement and fault reduction (Auvinen et al. 2005), one expectation was that the number of defects would decrease as a result of taking the agile practices into use. The results seem to support this assumption as well. The question is, are all of the defects really reported? Based on survey results, the answer is no, as some of the respondents said they did not report all of the found defects with the defect reporting tool. Thus, the reduction in the defect counts can be partially a result of managing defects somewhere else. That is also according to lean and agile practices (Poppendieck and Poppendieck 2007), as bugs should be fixed as soon as they emerge. Since this is a large organization with multiple scrum teams and legacy code, defects still do exist and get reported. Under these circumstances, a reduction in the number of open defects might not directly indicate an improvement in code quality while adopting agile. But it does indicate that there may have been a change to more flexible defect management practices and that, in turn, may have a positive impact on software code quality.

Flexibility comes from the fact that in agile, the scrum teams are usually co-located (Poppendieck and Poppendieck 2007), and thus it becomes easy to handle the defects online in the team's own code. Therefore, one might expect that the teams would report fewer defects to their own code via the official defect reporting tool. But in this study, the result was the opposite: teams started to report more defects to their own team with the official defect reporting tool. The Organization was novice in agility, so some defects could have been caused by inexperienced personnel, or the defect reporting tool might have been used as a defect backlog management tool (Korhonen 2009). Additionally, as the end result from the sprint should be good quality, potentially shippable software, (Poppendieck and Poppendieck 2007), the teams are using more time and test automation for testing their own code. Thus, more defects are uncovered and reported, as there is no time to fix all bugs at once.

As the development cycle is fast in agile, and both development and testing are ongoing at the same time (Crispin and Gregory 2009), one reason for using the defect reporting tool is to record time-consuming, complicated defects and logs of the problem situation in the tool for further analysis. Would it also mean then that the more simple faults are fixed as fast as they arrive, and there would be fewer minor-class defects reported in the agile software development? The results in this study did not indicate any major changes in the numbers of reported defects in different severity classes between the projects.

12.5.3 Proposed measures

As an outcome of this study, it is proposed that large software development organizations would follow up the changes in the defect data and defect reporting practices as well as people motivation during the first year of their agile transformation. Some examples of topics to be checked from the data:

- Is the total number of open defects reduced over time?

- Is defect lifetime getting closer to sprint length?
- Are there changes in the defect reporting practices within the teams?
- Are these changes in line with the expectations?
- Is it clear to everyone how to handle defects in agile?
- Is there guidance on how to do the prioritization between defect fixing and content development?
- Is there improvement in people motivation and satisfaction after the change?

In addition to this, it is recommended that the organization analyzes whether the agile methods have lead to better management measures, faster development time, and more releases. The analysis provides information on the changes, which can help in assessing if the impact of agile practices is what was expected and if there is a need for further actions to improve the situation.

12.5.4 Reservations

The impact of the agile transformation can differ from the results in this article for a few reasons. One important issue to consider is the procedure the agile practices are adopted within the organization. The Organization in this study started from practices that are more focused on team practices. Starting with coding practices, such as TDD (Nagappan et al. 2008), the impact on defect data probably is more imminent. On the other hand, this study shows that adopting the team practices already can have a positive impact on the results.

Other issues are the size of the organization and the length of the production cycle. The smaller the organization, the easier and faster the agile adoption usually is--especially if the development is co-located. Faster production cycles provide a faster feedback loop to development teams, and people can learn and become more experienced with agile practices during the first six months. But as shown in this study, even in a large company it is possible to get a positive impact during the first six months of agile adoption.

Finally, people's experience with software development in an organization may have an impact on the results. In this study, most of the people in the Organization had lots of experience in traditional software development. Sometimes breaking the old way of working, the "follow the plan" approach in traditional software development (Poppendieck and Poppendieck 2007) can take time, resulting as a lack of visibility to the actual change, for example, in the defect data. By providing feedback, which is based on real defect data and collecting input via surveys, it may be possible to support people in their agile practices' adoption (Korhonen 2010).

12.6 Conclusions

This study was conducted in a large, multisite organization during the first 12 months of its agile transformation. The goal was to explore the impact of agile on the defect data and defect reporting practices. The analysis was based on interviews before the transformation, the defect data records, and two consecutive surveys made in the project organization. The results showed that defect reporting practices changed, resulting as improvement, for example, in closing of the defects. Results also indicated that there can be challenges in the migration of the defect management to agile.

Based on this study, it is proposed that organizations, while starting their agile transformation, should keep track of the changes in defect management to follow up if the changes are consistent with what is expected. The key practices to improve defect management are:

1) product backlog to prioritize the work;

2) the continuous integration with a proper set of automated test cases; and

3) a short sprint cycle.

The last two are needed to provide fast feedback loop for the problems in the system-level software build.

In future research, the plan is to study which of the adopted practices remain in use, and if some of the practices are discontinued, what are the reasons behind it. Based on this information, further analysis will be done on the long-term impact of these practices on the defect data, and on quality improvement in the software development.

13 ADOPTING AGILE PRACTICES IN TEAMS WITH NO DIRECT PROGRAMMING RESPONSIBILITY

Originally published as

Korhonen, K.(2011): Adopting Agile Practices in Teams with No Direct Programming Responsibility – a Case Study. Proc. 12th International Conference on Product-Focused Software Process Improvement (PROFES 2011). Italy. LNCS 6759, Springer, pp. 30-43.

14 EVALUATING THE IMPACT OF THE AGILE TRANSFORMATION

Submitted for publication in Software Quality Journal as

Korhonen, K. (submitted 2011): Evaluating the Impact of the Agile Transformation – A Longitudinal Case Study in Distributed Context

14.1 Introduction

The findings of most studies on agile benefits suggest that agile practices improve quality if implemented correctly (Sfetsos and Stamelos 2010) and that the agile practices bring added value in comparison to the plan-driven approach (Petersen and Wohlin 2010). Applying agile practices in the organization is expected to result in better, cheaper and faster software development, responding to customer needs. The positive results and the expected benefits regarding flexibility in the changing business environment and the possibility to provide real value to the customer have encouraged organizations to take agile practices into use. But how does a company know when enough agility has been reached, and how do they know they succeeded in their transformation?

Although adopting the agile practices is an important topic in software engineering, only few in-depth studies focused on the impact of the organization wide agile transformation from defect management point of view. Petersen and Wohlin (2010) provide a comparison between plan-driven and incremental/agile approaches concluding that incremental and agile practices bring added value e.g. by reduced fault slip through. A recent study from a large-scale, companywide agile transformation by Laanti et al. (2011) concludes that most respondents agreed on all accounts with the generally claimed benefits of agile methods, including increased quality and earlier detection of defects. However, a study by Li et al (2010) did not show a significant reduction of defect densities or changes in defect profiles after transition from plan-driven process to scrum. As nowadays many software companies are changing from plan driven software development to agile, it is important to perform further empirical studies to provide better understanding on the impact of agile transformation on the software quality, and from defect management point of view in particular.

In this chapter, the impact of the agile transformation is evaluated from defect management point of view against the goals which were set for the transformation by the management before the agile transformation was started. The analysis is based on 1) qualitative data, which was collected from the personnel in the organization by two surveys on the goals and

practices, and 2) quantitative data, which was collected from the defect records. The defect data analysis, analysis on the practices to support reaching the goals, and survey results on the impact of agile in the daily work practices are presented in this chapter.

This study focused on evaluating the impact of agile on perceived improvement in four areas: visibility to what is being done, capability to react, quality of software development and motivation of the people. The results show that agile practice adoption had a positive impact on all these four areas though there were some challenges faced as well.

The remainder of the chapter is structured as follows: Section 14.2 describes the background for this study. Research setup is presented in Section 14.3, and the empirical results in Section 14.4. The chapter is concluded with an analysis on the impact of the agile transformation in Section 14.5, and final remarks are presented in Section 14.6.

14.2 Background

14.2.1 Agile transformation

Kettunen and Laanti (2007) propose that before starting the agile transformation, the company needs to make a strategic choice to select what kind of agility the company needs. The principal goals are suggested to be 1) being fast and responsive, 2) achieving high development productivity and 3) creating products with distinction and integrity. Different companies, and even different organizations within a same company, can value the goals differently, and there are many possible ways to achieve the goals. In order to implement the goals of the agile software development, many enabling factors, such as appropriately resourced, trained and empowered project teams, need to be in place as well.

In large organizations, getting to the fully agile level can take a few years, and patience is required. A survey made at Nokia Siemens Networks (Vilkki 2009) concludes that several agile engineering practices as well as a sustainable pace need to be in place before significant improvements e.g. on code quality can be reached. In this study, "basic" agile practices included short time-boxed iterations, product backlog, continuous integration, retrospectives and self-organizing teams. "Intermediate" agile practices add to basic practices refactoring, TDD, or acceptance test driven development (ATDD), or tests written at the same time as code. More practices than that would be equated to a "fully agile" organization.

One of the building blocks of motivation in agile transformation is a sense of progress. Visibility into the progress reinforces motivation, and is one way to empower the team (Poppendieck and Poppendieck 2007). But it can be difficult to measure your success against a predetermined plan because the plan can change in every sprint. Working software itself should be the primary measure of progress, and failures in the required behavior of the software system, software defects, can be used to provide feedback on the progress. Therefore, one simple way to provide feedback on agile transformation progress is to

compare the measures before and after agile transformation (Ileva, et al. 2004). There are defect-related quality metrics introduced in agile studies; for example, the number of customer-reported defects in a specified time following the release (Schatz and Abdelshafi 2005), the number of defects by iteration (Ileva et al 2004), enhancements made per defects reported (Poppendieck and Poppendieck 2007), effort spent on bug fixing (Ileva et al. 2004), and defects carried over to the next iteration (Hartmann. and Dymond 2006).

14.2.2 Expected impact of agile transformation

Being fast and responsive means that the company is able to respond more flexibly to the market changes, and deliver new features fast. To achieve this, agile methods propose to work in short (max 4 weeks), time-boxed iterations (Schwaber and Beedle 2002), where the customer involvement is mandatory both in the beginning and at the end to provide timely feedback both on the requirements and the end result. This makes it easier to implement the changes fast, be more responsive to the customer needs and thus improve the customer satisfaction (Ceschi et al. 2005, Sillitti et al. 2005). In fact, at Cisco (Lifshitz et al 2008) one of the problems listed during the agile transformation pilot was indeed the customer not being present to answer questions during the iteration.

Productivity can be improved by focusing the development work, and by using fast feedback loops. Test Driven Development, TDD) suggest writing the test cases first, and then doing the programming to pass the test case, which provides fast feedback on the design to the developer. Continuous integration is important in large organizations to provide fast feedback loop on the system level product.

Team capability, including expertise and motivation, is proposed to be one of the main success factors in agile software development (Poppendieck and Poppendieck 2007). It is also proposed that a certain percentage of experienced people are needed for a successful agile project (Lindvall et al. 2002), and the three most important success factors for agile transformation would be culture, people and communication.

It has been studied that there is a link between the perception of the change in the code quality, visibility to the change in the defect data, and emotional engagement to agile transformation (Korhonen 2010). In the agile pilot at Ericsson (Auvinen et al.2005), team members found practices like pair programming as highly motivating, and they enjoyed the challenge of new things and learning. On the other hand, it has also been reported that more experienced engineers had motivation issues during agile transformation (Lifshitz et al 2008). At Nokia (Laanti 2010), the positive attitude towards the agile practices within the organization had an impact on the newcomers, who were positive on agile even though they had not yet worked with it.

In nowadays business, product distinction from other vendors, and the integrity of the product are essential. There are studies reporting that implementing agile practices have helped to

improve code quality. For example, at Primavera (Schatz. and Abdelshafi 2005), agile adoption increased their product quality by 30%, and after 7 sprints of implementing TDD, the number of defects was reduced by 10% per team. At Cisco (Lifshitz, et al 2008), members of the pilot team adopting agile practices felt that product quality was higher, and they were consistently delivering at the end of each iteration. At Yahoo! (Benefield 2008), feedback on agile transformation was collected from the teams, and 54% of the respondents felt that agile practices improved the overall quality and "rightness" of what the team produced. At Nokia (Laanti et al. 2011), the study results revealed that, after the agile transformation most respondents agreed on all accounts with the generally claimed benefits of agile methods. These benefits included higher satisfaction, a feeling of effectiveness, increased quality and transparency, increased autonomy and happiness, and earlier detection of defects.

The impact of all the new practices cannot be expected to become visible immediately. A three-month long agile pilot project at Ericsson (Auvinen et al. 2005) resulted in a 5.5% decrease in defects, but formal code review revealed that the quality of the code was the same as previously. On the other hand, another study at Ericsson (Petersen and Wohlin 2010) revealed that the critical issues encountered in plan-driven development were no longer issues after implementing the agile and incremental practices, which implies that agile practices bring added value in comparison to the plan-driven approach - though Agile adoption can bring other problems. Li et al. (2010) reported in their longitudinal study on transformation from a plan-driven process to scrum that the software quality and knowledge sharing got more focus when using Scrum. However, they also found that Scrum put more time pressure and stress on the developers which made them reluctant to perform maintenance related tasks.

14.3 Research setup

The research in this study was done as an industrial case study (Yin 1994). The context of the research, research questions and data collection is presented next.

14.3.1 Context

The organization in this study is developing telecommunications software with over 150 experts working globally. The organization had several years of experience on developing the software with traditional, plan-based methods before they started the agile transformation. This study was done during the one year agile transformation period.

Based on the analysis of the current issues, and expected outcomes of the agile transformation, the following four goals were set for the transformation by the managers before the transformation started:

Goal 1) Increase visibility: we are able to provide better visibility to what is being done.

Goal 2) Increase capability: we are more capable, and faster in reacting to changes.

Goal 3) Improve quality: the quality of software development improves.

Goal 4) Motivate people: the personnel in the organization are motivated to work in the new way.

To start with the transformation, a set of agile practices was selected and implemented organization wide. These practices included short time-boxed iterations, product backlogs, continuous integration and self-organizing teams. In addition to these practices, the teams with direct code development responsibility were optionally taking into use more technical, code development related practices such as TDD, refactoring, tests written same time as code and pair programming. Table 17 summarizes how these practices were assumed to support the defined four agile transformation goals.

Table 17 Agile transformation goals and which practices mostly support reaching the goals (marked with "x").

	Goal 1: Increase visibility	Goal 2: Increase capability	Goal 3: Improve quality	Goal 4. Motivate people
Time-boxed iterations	x	x		
Product backlogs	x	x	x	
Continuous integration	x	x	x	x
Self-organizing teams		x		x
Technical practices			x	

The short, time-boxed iterations increase the capability to react, as the decisions on the content can be done after each sprint for the next few weeks (Goal 2). Demo at the end of the sprint provides fast feedback cycle improving the visibility on the items which were done (Goal 1). Product backlog makes the prioritization of the content items visible (Goal 1) and it is easier to adapt to the needs of the end user (Goal 2 and Goal 3). Fast feedback cycle is available via continuous integration, which supports all four goals. Self-organizing teams is said to motivate people, as the scrum team has the responsibility on the tasks they can commit to during the sprint. When the team is fully agile, i.e. a self-organizing, cross-functional team, it will ultimately improve the capability to react as the team has the needed competencies and capability to react within the team. Introducing the technical software development practices such as TDD, improves the software development quality, and finally also the quality of the end product.

The agile transformation was facilitated during the first year by an agile transformation team, which consisted of managers from the organization, and a couple of external consultants. In the very initial phase, several introductory presentations were held for the personnel by international well-known agile professionals. Later on workshops were arranged on different agile topics by external speakers, and the majority of the personnel participated in the Certified Scrum Master training. The external consultants were supporting the teams in the start up phase by facilitating e.g. Ways of working –workshops, where the team agreed on the agile working methods for the team. At the same time, internal group of agile coaches emerged to support the teams further.

14.3.2 Research questions

The main goal of the study in this chapter was to evaluate the impact of the agile transformation in the studied organization with respect to the goals set for the transformation, and by utilizing the defect data and feedback from personnel collected in this study. The following research questions were derived from the goals set for the agile transformation:

After the agile transformation was started,

- has the visibility improved?
- is the organization faster in reacting to the changes?
- has the quality of the software improved?
- are the people in the organization motivated to work in the new way?

Additionally, the data was analyzed to provide an answer whether the distributed organization had an impact on the agile transformation process.

14.3.3 Data collection and analysis

Three sources of data were used to analyze the impact of agile transformation: defect data records, survey responses, and comments from feedback sessions after the surveys. Defect data was collected from five successive software development projects before the agile transformation started. During the agile transformation, data was collected from three software development projects. The scope of all these eight projects was to develop the next version of the same, existing software product in the telecommunications domain.

To analyze the impact from defect data, two metrics were used: Number of open defects and Defect cycle time metrics. The detailed description of these metrics is provided in section 14.4.1. The combination of these two metrics provides visibility to the expected impact of agile practices on the defect management during the software development. For example, if the number of open defects is reducing, this improves the visibility of the remaining problems and may indicate improvement in software development quality as well as improved capability to react as the defects are handled without need to be reported with tools.

From the survey responses, first the progress for adopting the agile practices was analyzed, as certain agile practices were expected to have an impact on reaching the goals. Secondly, the analysis on survey responses to the Impact of agile to daily work - question was done to collect feedback on how the respondents perceived the impact of agile in flexibility to respond to changes, in the visibility of the actual status of development, in motivation and enthusiasm of team members, and in the quality of code. After the survey, there were feedback sessions arranged within the survey participants, and comments from the discussions there were recorded for this study. This data was utilized to evaluate the distributed development aspect as well.

14.4 Empirical results

14.4.1 Defect data

The change in software development is evaluated by the changes in the defect data management, as the defects are considered to be one indicator of software development quality. The goal is to analyse whether the adoption of agile practices had the desired impact based on the defect data and survey responses. This analysis is divided into two sections: Number of open defects and Defect cycle time. The metrics with the empirical results from this study are described in more detail in the following sections with related survey results.

14.4.1.1 Number of open defects

An open defect is a problem which has been found in the software but not yet corrected, or the correction has not yet been verified. This metric measures the number of open defects at a given point of time, and the data is collected on regular interval throughout the software development project.

The number of open defects graph can be used to visualize the readiness of the software for releasing – if there are many critical bugs reported but not yet corrected in the software, or the number of open defects is increasing over the time, the software maturity might not yet be ready for mass deliveries. The expectation is that the number of open defects will stabilize, and decrease to minimal before the release date.

In this study, the number of open defects was recorded on a weekly basis for each project several weeks before and after the software was frozen for the final system level integration and verification. The number of open defects at the measured day was compared to the maximum number of open defects measured during the first project, Project 1.

After Project 1 there were enhancements done to the plan based process: the defect management practices were improved, and integration testing was started earlier in the development phase. In Figure 27 the number of open defects over the time is presented for the waterfall Project 1 and the waterfall Projects 1a, 1b, 1c and 1d.

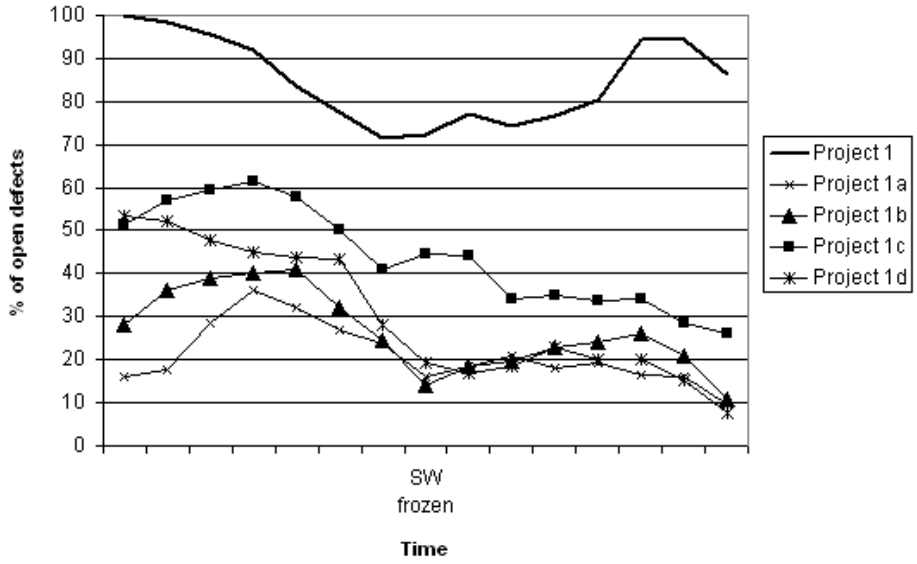

Figure 27. Number of open defects over time during the waterfall baseline project 1, and during the enhanced waterfall on-top projects 1a-1d.

There were two main findings. First, compared to Project 1, the number of open defects was constantly lower in projects 1a – 1d, which is mostly explained as the Project 1 as a baseline project was more complex, had more content and the development time was longer compared to the on-top projects. On the other hand, in project 1 there was a challenge with slow closing speed, and thus, open defects were accumulating. During projects 1a-1d, more attention was put on following up the defect closing speed as part of the defect management improvement actions, which might have had an impact on reducing the number of open defects in these on-top projects.

The second finding was that after introducing the enhancements, there was no such a high peak in the number of open defects during system testing as in Project 1. The high peak in Project 1 during the system testing phase was mainly caused by integration and installation problems as the installation was not tested until that point. The enhancement to start the integration testing already during the component testing phase clearly improved the situation, and those defects were handled earlier in the development. The trend line in project 1c is slightly different due to the fact it had one additional system testing round before the official system testing started and this revealed new defects while the software maturity was not yet on the level to be frozen for system testing.

Figure 28 presents the defect data for the on-top releases 2a, 2b, during which the agile transformation was started. Baseline project 2 was still developed in the plan based mode, so it is left out from this figure but used later on in the comparison between the baseline projects (Figure 29).

The main difference compared to Figure 27 is that after introducing the agile practices, the number of open defects was lower also *before* the software freeze point in the agile projects 2a and 2b than in the previous, plan based projects. The defects were reported with regular

speed over the time, and there were no high peaks in the number of open defects in the agile projects 2b and 2c before or after the software freeze point.

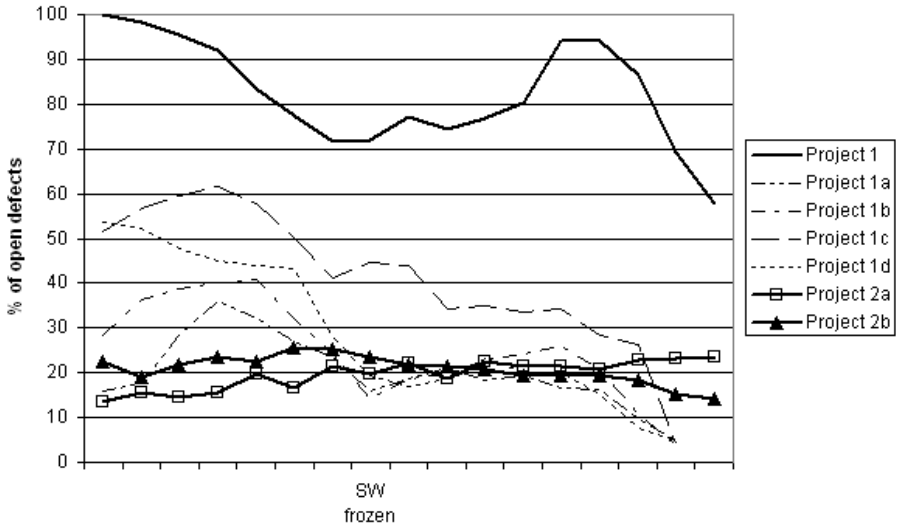

Figure 28. Number of open defects over time during agile transformation in the on-top projects 2a-2b. In the background are the lines for the Project 1 and on top projects 1a-1d for reference.

In Figure 29 the comparison is done separately for the baseline projects due to the different level of complexity in these projects compared to the on-top projects. Usually the baseline brings bigger, more fundamental changes, thus the complexity is greater and there are more changes made to the code. Therefore the testing needs to be more thorough and more defects may come up also during the later testing phases.

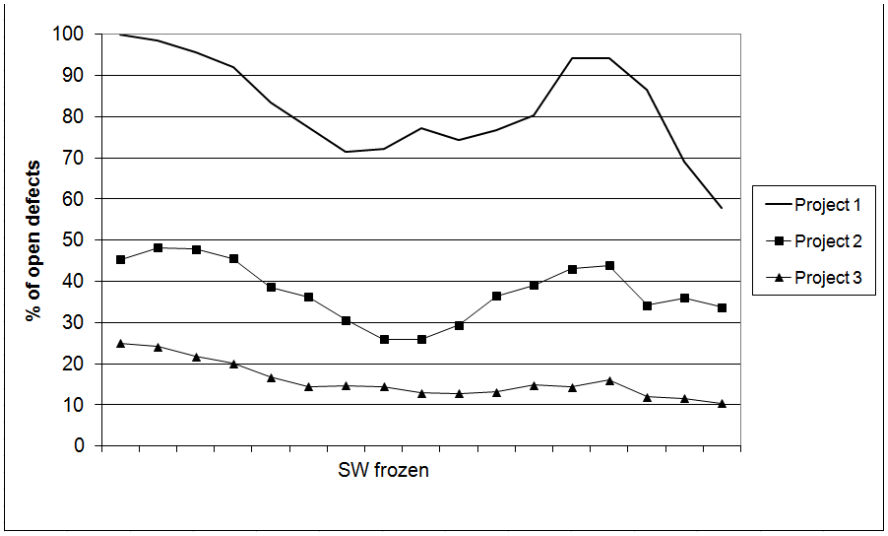

Figure 29. Number of open defects during the baseline projects Project 1(plan based), Project 2 (plan based with enhancements) and Project 3 (agile).

Project 2 still has a higher peak after software freeze point even though the earlier integration and defect management enhancements were in use, so the enhancements to the waterfall

mode did not have exactly the same impact than in the on top projects 1a-1d. On the other hand, the impact of agile practices adoption seems to be consistent also for the baseline projects. During the agile baseline project 3 the trend line for the number of open defects over the time is more flat and there are less open defects reported than in the previous baseline projects with traditional waterfall software development model (Project 1) or waterfall with improved defect management practices and earlier integration (Project 2).

The difference in size and complexity might be one reason to explain the decrease in the number of open defects, and adopting agile practices also is promised to bring better quality thus reduce the number of open defects. According to the agile principles, the defects should be handled as soon as they arrive, preferably within the same sprint. So one explanation can be that not all the defects even end up into the defect tracking tool during the agile development. In the survey, the respondents were asked whether they report all the defects they find with the defect tracking tool. After 6 months, 51.1% of the respondents reported all faults they found, 26.6% did not (Table 18). After 12 months, 80% of the respondents said they report, and 17.5% said they do not report all the defects with the reporting tool.

Table 18. The results to a survey question about reporting faults with the fault reporting tool.

Agile experience	I report all faults I find with the fault reporting tool			
	Agree	Not agree or disagree	Disagree	I do not know
6 months	51.1%	16.7%	26.6%	5.6%
12 months	80%	2.5%	17.5%	0%

Based on the discussion with the teams and the analysis on the responses to the open-ended questions, the comments were similar on why all the defects are not reported with the defect tracking tool. Respondents felt it unnecessary to report defects which they could fix right away in their team, or it was faster to get the fix by walking to or calling the developer in another scrum team directly. But still the tool was used, as it was easier to use the existing tool for defect reporting than start using some other tool or practice.

14.4.1.2 Defect cycle time

The Defect cycle time is measured from the date when the defect was found and reported in the defect reporting tool until the date when the correction was verified for the reported problem. This metric can be used to find early information on possible problems in either fixing the faults or verification of the corrections.

This metric was first implemented in the studied organization for the plan based waterfall projects to improve the defect management. At that time, a target time limit based on history

data was set for closing the defects, and the Defect cycle time was measured for each defect class by calculating how many of the defects were fixed and verified within the target time limit. Time was calculated in days including both correction time and verification time.

For the data collected during the agile transformation the metric was enhanced a bit to provide more valuable information. One of the goals in the agile software development is to fix the defects while they arrive, meaning that the defect correction would preferably take place during the same sprint it was found. Therefore, for the agile projects, the defect cycle times were compared to the sprint length.

To make the comparison between the different baseline projects in this study, the defect closing times from the waterfall projects were also compared to the same sprint length as the agile projects' defect closing time. The entire defect closing times where then compared to the Project 1 defect closing times. For the graphs, the defects have been grouped according to the lifetime into six cumulative groups: percentage (compared to Project 1) of defects which were not closed in time less than or equal to the length of half a sprint, one sprint, one and a half sprints, two sprints, two and a half sprints and three sprints. In Figure 30, the defect closing time is shown separately for each baseline, and on top releases.

In general, the closing speed seemed to be faster in the on-top projects when comparing to the respective baseline releases (Figure 30). For example, after three sprints, Project 2 had over 4% of defects (compared to total number of defects in Project 1) still open, and in projects 2a and 2b there were less than 2% of defects open after 3 sprints. Similar trend is visible with Project 3 and on top projects as well. In Project 1 and on top releases the difference is even greater, in Project 1 there were over 20% of defects not closed after 3 sprints, and thus it is not visible in the figure. In the on-top releases, there were 4 % or less defects open after 3 sprints. This difference between on-top and baseline releases may be related to fact that the changes done in the baseline are more complex, and thus the fixing of the reported bugs also takes more time than in the on-top releases.

When comparing the baseline projects (Figure 31), the defect closing speed is getting faster project by project. Additionally, the number of defects open less time than half of the sprint has decreased from 25% in Project 2 to 11% in Project 3.

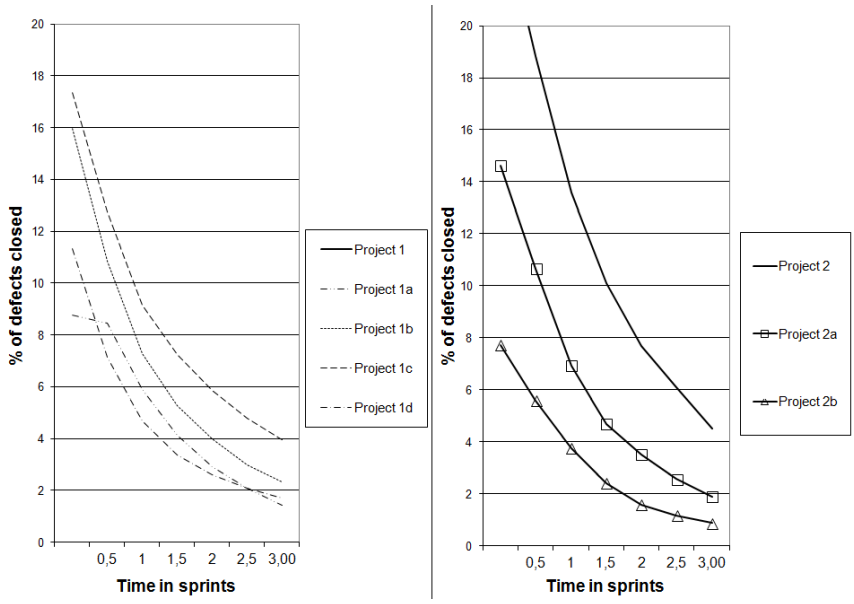

Figure 30. Comparison of the closing times between the baseline (solid black line) and on-top releases.

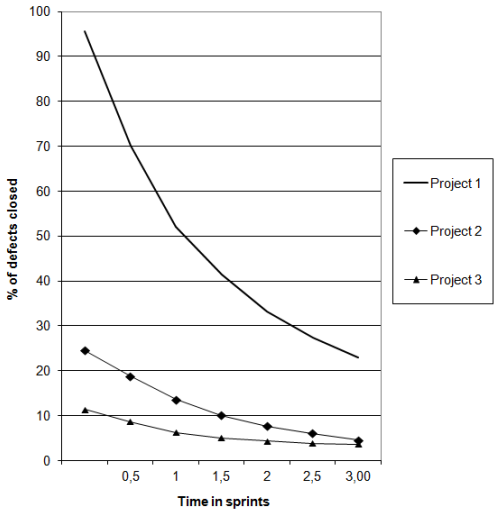

Figure 31. Comparison of the defect closing time between baseline projects 1, 2 and 3.

14.4.2 Practices

To illustrate the change in the agile experience during the transformation, the experience is presented with practices in use (Figure 31) based on the survey responses 6 months and 12 months after the agile transformation was started. During the first year of agile transformation, the usage of the four, non-technical practices was increasing clearly. After 12 months, the time-boxed iterations were used by nearly all (98%). Product backlog was in use by 97% of the respondents, and they felt that the teams were mostly self-organizing (93 %).

Also the continuous integration utilization had increased from 35% to 75% during the latter 6 months.

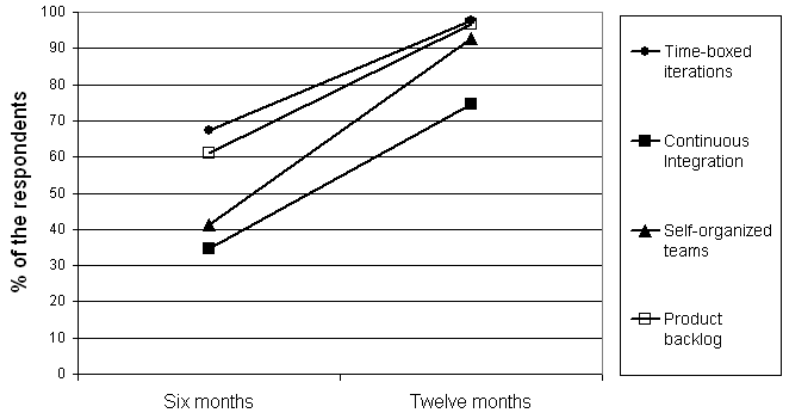

Figure 32. Change in agile practices in the whole studied organization.

At the same time, the programmers were taking the more technical, code development related practices into use. In Figure 33 it can be seen that code refactoring was used (50%), and tests were written at the same time as code by 50% of the respondents after 12 months of the transformation. Pair-programming and TDD were not in use that much, but still the usage had increased from less than 10% to over 30% during the latter 12 months after starting the agile transformation. 100% utilization of these practices was not expected, as the group of respondents included also people with no direct programming responsibility.

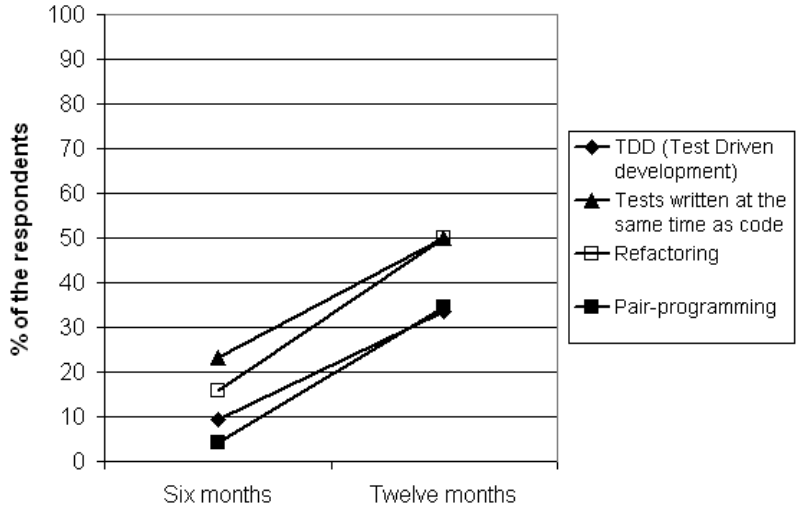

Figure 33. Change in the code development practices in the studied organization

14.4.3 Impact of agile to daily work

After half a year of starting the agile transformation, the survey results were quite positive about the change (Table 19). The motivation had improved (according to 53, 2% of the respondents), as well as the flexibility to respond to changes (67%) and visibility to actual

status of development (72%). Even when the agile adoption was at quite early stage, some respondents felt that the quality of the code had improved (23.2%) as well as team work between different locations (54.7%).

Table 19. Survey results after half a year of starting the agile transformation to the question "*what are the impacts of agile development compared to traditional development*".

	In your experience, what are the impacts of agile development compared to traditional development?			
	Improved	No impact	Gotten worse	I do not know
Flexibility, ability to respond to changes	67%	18.1%	3.2%	10.8.%
Visibility to actual status of development	72%	10.8%	6.5%	10.8%
Motivation and enthusiasm of team members	53.2%	27.7%	9.6%	9.6%
Quality of code	23.2%	23.2%	23.2%	30.5%
Team work, communication and cooperation of teams in different locations	54.7%	30.5%	6.3%	8.4%

There were also respondents who felt that the quality had gotten worse (23.3%). Based on the responses to the open ended questions, and feedback discussions, some respondents were concerned on the missing code ownership. Some of the programmers felt that the code was getting worse as too many people were working on the same code. This also caused a drop in the motivation. Lack of long term planning was also mentioned by some respondents as one reason behind the 9,6% of people feeling that the motivation had gotten worse within the team. Some respondents felt that, as the big plan was missing, the short term goals were not consistent from one sprint to the next, and resulted as bad code quality. One issue was also the feeling that the agile practices could not be applied to specific work which was not directly programming related, and it was for example unclear how to do testing in agile. The unclarity of how to actually adopt the practices caused frustration among some respondents.

Quite many persons, 30,5% of the respondents, felt that they did know what kind of impact the agile adoption had on the code quality. Analysis of the survey responses brought up two possible explanations: majority of these people did not have direct programming responsibility, or they did not have visibility to the changes, e.g., in the defect data metrics. Feedback discussions revealed that the people felt that they did not yet have enough evidence about the possible impacts on the code quality, and, consequently, they were not able to share their opinion on the impact of agile in the code quality.

In the team work, 30.5% of the respondents felt that the agile transformation did not have any impact on the co-operation between the different locations. In the feedback session there were comments that the organization had been distributed already before the agile transformation, and there already existed communication channels between different locations. Few respondents felt that the communication had gotten worse, and one comment explaining this in the survey open-ended responses was that " *all my scrum team members except me is located on the other site, it is difficult to participate to the daily scrum meetings over the teleconference line*".

14.5 Evaluating the impact

The goal of this study was to collect real data on the agile transformation impact on a large, distributed organization, and assess the impact of agile transformation with respect to the goals set for the transformation by the management. In the following the results are discussed separately for each of the goals.

The first goal for the agile transformation was to provide better visibility to what is being done, and results show that this had improved clearly. After 12 months of starting the agile adoption, both product backlogs and time-boxed iterations were used by over 97% of the respondents. Product backlogs and time-boxed iterations provide clear goals, fast feedback loop, and therefore improve the visibility of what is being done. Responses to the survey confirm that there was improvement, as 72% of the people felt that the visibility had improved during the first 6 months of the agile transformation. Additionally the decrease in the number of open defects improved the visibility to the biggest problems since the more time-consuming, and more complex defects were reported in the tool. The continuous integration on the other hand was in use according to 75% of the respondents, so there is place for improvement still to provide better visibility.

The second goal for the agile transformation was to increase capability to react, and the results provide evidence on improvement in this area as well. Time-boxed iterations provided a chance to change the plans fast, and product backlogs helped to prioritize the content to be developed in the sprints. Self-organizing teams allowed the teams to take more responsibility on what is being done and how, and made it easier to focus on the tasks at hand within the team. All these three practices were used by over 90% of the respondents after twelve months of starting the agile transformation. The extent of adoption of these three practices would indicate that the capability to react had improved significantly. This was noted by the personnel as well, since 67% of the respondents felt there had been improvement in flexibility and ability to respond to changes. It was expected that the capability to react to defects would improve as well, thus the defect closing time would improve over time. The defect data metrics showed improvement in this area. Further analysis revealed that this would be a result of fixing the easy, "fast", defects as soon as they arrive. Thus they are not even reported into the tools, and only the more time consuming or complicated defects were

being reported with the tool. This was also concluded by the respondents, as not all the defects got reported with the tool by all the respondents.

The third goal for the agile transformation was to improve the quality of software development. Based on the survey responses, 23,3 % of the respondents felt that the code quality had improved. All the applied practices support reaching this goal as well, so the impact was further analyzed from the defect data. Based on the results, the number of defects had reduced and the defects were reported on regular speed over the time . This was an expected result based on the previous reports. Explanation to this seems to be again that not all the defects get reported into the defect tracking tool. Also, this can be a result of more test automation, and defects handled elsewhere than in the defect tracking tool. One concern though is possible lack of time for running all the testing – at least some respondents felt that the short time-boxed iterations made more pressure to make same testing but in a shorter time. This pressure, and the worry of missing code ownership were some examples of why 23,2 % of respondents felt that the code quality had gotten worse. Additionally, the programming related practices, TDD, refactoring, pair programming and tests written same time as code, which would help to improve the quality, were not yet that widely in use among the developers. Getting more of these practices into use would be needed before quality of the software development would improve further (Vilkki 2009).

Finally, the fourth goal for the agile transformation was to motivate people. Continuous integration provides fast feedback for the team, thus supporting the feeling of progress and therefore motivating people (Poppendieck and Poppendieck 2007). The self-organizing teams is the basis for team work, responsibility taking and achievement, which indeed should support the motivation if successfully implemented within the team. This was seen also from the responses in the survey: the majority of the people, 53,2 % of the respondents, felt that the motivation and enthusiasm of team members had improved after agile transformation started. But there were also people who felt that the motivation had decreased (9,3%) or that there had not been any impact (27,7%). This may be a result of missing long term plans, or uncertainty how to work in the self-organizing team and adopt the agile practices as commented by the respondents during the feedback discussions. The change from a plan-driven development to being self-organized, responsibility taking team is quite huge for the team, and it depends on the individuals how well they can adapt to the new way of working.

As a conclusion, the collected feedback, and analysis on the defect data shows that there was clear progress towards reaching all of the goals within the studied organization. Especially the visibility to what is being done, and capability to react improved according to all data. One area to improve for both of these goals would be the continuous integration. The third goal, improving the quality of the software development had improved as well, but there was more room to develop this area further. While the agile practice adoption, especially the code development related practices, increases, it may improve the quality of software development further. The uncertainty of how to adopt the agile practices, and feeling of bigger pressure to do things in shorter time decreased the motivation. Majority of the people were more motivated than during the previous way of working, but still there were plenty of people who

either were less motivated, or did not see any change in the motivation levels. Possibly, as the agile implementation brings the problems at hand more clearly visible, people may feel that things are getting worse, decreasing motivation. At this point it would be recommended to provide feedback on the actual progress of the agile transformation to the people, to support them in their transformation process as discussed in chapter 13 of this dissertation.

The distributed environment did not seem to have any major impact on the agile transformation progress. This can be explained with the fact that the organization had already got familiar with the large scale software development, and the agile practice adoption did not bring any big changes that would have changed the way of working between the locations. The new way of working was communicated at every site with the same professionals, and the general guidelines were same for each of the teams. In fact it seems that through the continuous integration and common product backlog the feedback, information sharing and communication between the different locations improved. The fast feedback cycle, and working in the same sprint cycle helped the teams to align their activities, though there is still place for improvement.

The results in this study focus on the impact of the transformation within the studied organization. To analyze the impact of agile transformation further, the end user view could be included. Additional data could be collected from the customers, and how they perceived the change of the agile practice adoption in the end product. Two topics in addition to customer interviews could be evaluated to get feedback on the impact: the feature cycle time, and defects reported by the customers. The feature cycle time measures the feature development time from the original customer request to actual delivery date. If this time is getting shorter, then the organization can claim that agile practices have ultimately made the development cycle faster. The defects reported by the customer after releasing the software can be used to evaluate the impact of agile adoption on the product quality. If the number of customer reported defects reduces release by release after adopting the agile practices, the organization can claim that the quality of the product has improved. This data was not collected in this study, but it should be in the scope of future study to provide a complete view on the impact of the agile transformation.

14.6 Conclusions

The main contribution of this study is to provide real data of the agile transformation impact in distributed context. Based on the results the adoption of short time-boxed iterations, product backlogs, continuous integration and self-organizing teams improved visibility to what was being developed, improved capability to react and motivated people. The application of agile practices was also visible in the defect data, as the number of open defects reduced, and the defects were reported on regular speed over time in the agile projects compared to the traditional projects. This alone does not directly imply an improvement in the code quality, but it can be seen as an indicator of quality improvement in the way of

working. Improvement in the code quality takes time, and requires organization wide usage of agile practices, such as continuous integration. In future this study will be continued with further analysis to get more feedback on the impact of the agile transformation on the code quality, and on the quality of the software development in general.

This study can be used as a reference for other companies on how to evaluate the impact of the agile transformation in their organization, and thus support the personnel in their transformation. Early indicators on the impact of the selected agile practices can be used to evaluate whether the transformation is going to the direction that was originally planned, and this analysis gives the organization a chance to align the activities accordingly where needed.

Part V Closure

In this part, a summary of the contributions and a discussion on the limitations of this study are provided. Implications for practitioners are discussed, and the part is concluded with final remarks.

15 REVIEWING THE CONTRIBUTIONS

The research questions and the contributions of the study are reviewed in this section.

1. Which quality metrics support the defect management process in a large, distributed organization?

Three metrics were selected with the GQM method for this study: Total number of open defects, Defect finding profile and Defect lifetime. According to the results, the combination of these three quality metrics helped to support the defect management process in multisite software development programs: the defect closing speed was improved, the number of open defects was reduced, and defects were reported earlier in programs that were using the selected three quality metrics.

2. What kinds of challenges are there in defect management during the agile transformation?

Three software development organizations were studied, and five issues related to processes, tools and metrics were identified as potential challenges in migrating defect management to agile in a multisite organization. The proposed actions are respectively: 1) create guidelines on what faults need to be reported, 2) evaluate the appropriateness of the fault reporting tool and related practices, 3) establish a common process on how to prioritize between fault fixing and feature development, 4) select the metrics to be used in agile, and 5) make sure that communication practices are well established between distributed teams.

3. What kind of changes, if any, does agile practice adoption induce in the defect data and defect reporting practices ?

The results indicate that when the defect reporting practices changed after the agile adoption was started, the defect inflow was more stable and the defect closing speed improved. This provides organizations an approach for evaluating the progress of agile transformation on the basis of defect data and defect reporting practices in a large and distributed organization context.

4. What is the impact of visibility (or lack of visibility) to progress on the engagement to the agile transformation ?

According to the results, improvements were visible in the defect data, but fewer than 25% of the people in the organization felt that quality had improved. Further study revealed that a realistic perception of the positive changes in the defect data coincided with positive emotional engagement in agile transformation. Especially in large organizations, the visibility to improvements in the early stages of agile transformation is vital, because the lack of

visibility may result in motivation decrease among the people in the organization, thus slowing down the agile transformation progress. This result provides organizations with an example approach to analyze the impact of the progress visibility on the engagement levels during the agile transformation. Based on the analysis, the organization can decide whether actions are needed to further support personnel in their organization during the change.

5. What is the impact of the agile transformation in different work roles?

The results suggest that, as expected, the developer teams had adopted a wider range of practices but the agile adoption was progressing also in the teams with no direct programming responsibility. This is important aspect to consider in large organizations, where there can be different work roles and tasks which do not directly include programming. To get the full benefit of agile, it is usually proposed that the agile practices should be adopted organization-wide. The results suggest that even teams with no direct programming responsibility benefit from certain agile practices, which contributes to creating a positive attitude towards agile transformation organization-wide.

6. How the defect management data can be used to provide feedback on the impact of the agile transformation?

The result shows that by analyzing the practices in use, by exploring the defect data, and by collecting feedback from the personnel, it is possible to evaluate the success in reaching the agile transformation goals in a large and distributed organization already at a very early stage. This in turn will help the organization to identify potential problem areas at an early phase, and to adjust their activities accordingly to steer the agile transformation into the desired direction.

16 LIMITATIONS OF THIS STUDY

For various reasons, there are reservations for the results of this thesis. In the following, the potential limitations of this research are summarized. The limitations are discussed from two viewpoints: the organizational environment in which the research was conducted, and the research methods used in this work.

16.1 Environment

The results are mainly from one organization within one company which might impose a threat to the validity of the results in other product development organizations. However, Nokia Siemens Networks is a very large environment, containing different software development cultures inside one company. There are commonly agreed ways of working, but different locations and cultural backgrounds create variation in the ways of working. Therefore the results can be seen, to some extent, as results from several companies inside a large company consortium.

While adopting agile methods, both the size of the organization and the length of the production cycle may have an impact on the results. The smaller the organization, the easier and faster the agile adoption usually is – especially if the development organization is co-located. Thus results such as those described in this study can be visible earlier than after half a year of starting the agile adoption. Again, if the production cycle of the organization is faster than what it was in this study, the feedback loop to development teams is faster; people can learn faster and become more experienced with agile practices during the first six months. This, in turn, can result in changes in defect data and defect management practices sooner than in a large organization with a slower production cycle. But as shown in this study, even in a large company it is possible to get a positive impact already during the first six months of agile adoption.

The personnel's experience on software development in the organization may have an impact on the results. In this study, most of the people in the organization had a long experience in traditional software development. Sometimes breaking the old way of working can take time, especially because of the "follow-the-plan" approach in the traditional software development (Poppendieck and Poppendieck 2007). This may delay the agile adoption process. On the other hand, if the organization consists of people with previous experience on software development with agile methods, the change can be faster, and quality improvement more imminent. Therefore this study can be used as one reference for organizations planning their agile transformation.

One important issue to consider is the procedure with which the agile practices are adopted within the organization. The organization in this study started with more focus on the team practices than on coding practices. If the first thing which is introduced to the organization is related to coding practices, e.g. pair programming or TDD, the impact on defect data would probably be more imminent. On the other hand, this study shows that adopting team practices can already have a positive impact on the results.

Interfacing with non-agile teams and adopting agile practices in the business partner organizations were excluded from this study. In general, all the teams in the Studied Organization were utilizing agile practices, and business partners participated in the work in the role of Scrum team members.

16.2 Research method

Large-scale agile adoptions are hard to study in isolation (Runerson and Höst 2009), and in industrial setup an agile transformation within an organization is a one-time phenomenon. In this study, the research period was quite long, altogether over three years. During that long time the business situation undoubtedly evolves, and may bring a need to e.g. find people in new locations to work in the project, evaluate the business case for the product or do more or less subcontracting. For the product in this study the impact of these business changes was minor. Subcontracting was already in use, development was done in multiple locations, and the product had already been on the market for some time to validate its business case. The case study provided a natural empirical method to study the agile transformation in this thesis by comparing the data before and after the agile transformation.

To validate the results of the case study, data for the research was collected from several sources to follow the principle of triangulation. Quantitative data for the thesis was collected from defect data records. Interviews and two surveys were used as qualitative data to identify confirmative and contradicting information to the quantitative data. Additional information was gathered from project documentation reviews, and feedback discussion sessions with the survey respondents.

Defect data was collected from only one product over several years. To validate the results, a thorough literature study was done to compare these results with those from other companies. The literature study revealed that the decrease in the number of reported defects during the agile transformation is in line with the previously reported results from agile transformation in other software companies (Schatz and Abdelshafi, 2005, Sfetsos and Stamelos 2010).

A majority of the survey results are based on opinions, which as such do not provide clear facts on the progress and changes due to agile transformation. On the other hand, opinions of the personnel involved in the transformation cannot be ignored. Even if all metrics would

provide evidence on a successful agile transformation, it is not really a success if the people in the organization do not see the change and are not satisfied with the results.

For the survey it was not possible to collect responses from the whole organization. The organization is large, and distributed over several locations globally, so some selection had to be made. Some parts of the organization were new, and people in those locations had not been working previously in the waterfall projects included in this study. Therefore, those locations were left out, and participants for the survey were selected from locations where the organization had been stable for a longer period. A second criterion for selecting the participants for the survey was that they had experience on the traditional software project, and were working with the same product in the projects under study during the agile transformation. That way they were able to make a comparison between the old and new ways of working.

In a study like this it is possible that the subjects modify their behavior in response to the fact that they are being studied (so-called Hawthorne effect). To reduce the effect in this study, the surveys were conducted anonymously. Based on the survey responses and feedback discussions, the respondents also felt that the survey was a feedback channel rather than an evaluation on their behaviour.

17 IMPLICATIONS FOR PRACTITIONERS

The success of the agile transformation depends on how the people in the organization react to it, and if they see the benefit of the change. Therefore communication on the goals and visibility to the progress are important, and may impact on the engagement levels and motivation throughout the organization.

Defect data can be used to analyse the progress of the change as shown in this study. When planning the agile transformation implementation strategy, an important step would be to analyze how the planned actions could affect the defect management practices, and what potential challenges there might be in transferring the defect management practices to agile software development. Changes in defect management practices will have an impact on the collected defect data, and therefore on the interpretation of the changes in the metrics before and after the agile transformation. For example, in this study the number of open defects was collected for the software development project before and during the agile transformation. The change in this metric provided empirical evidence of positive change in the software development methods.

In a large organization the change may take long, and initial experiences as well as progress may differ in different parts of the organization. Collecting feedback from various sources, e.g. on how the people perceive the change in quality, and how the agile adoption is proceeding within the organization, will provide further information for analysing the change progress and to support the defect data analysis. In this study, for example, two surveys with feedback discussions were conducted to collect data on these issues, and to make action plans accordingly.

The goal definitions, agile practice selection and defect metrics are tightly interconnected. Based on the findings from this study, there are several recommendations for practitioners for issues to be considered when providing support for agile transformation with defect management. These findings are listed in Table 20.

Table 20. Recommendations for practitioners

Topic	**Related issues to be considered**
Defining the goal for the transformation	• Clarify what is the real goal you want to achieve. • Communicate the goal clearly throughout the organization. • Plan how you can follow the progress, and how you can evaluate the success in reaching the goal. Communicate this to the organization as well.
Selecting the practices	• Analyse the different agile practices to find out which would help in reaching the goal.

	• Create a deployment plan for the selected practices. Share also the reasoning why these practices would support the goal. • Make a realistic expectation for practice adoption – not all the teams can or even should adopt all the practices.
Selecting the defect metrics	• Check if reference data is available from non-agile projects to make the comparison. • Analyse what it is that you want to know, and what defect metrics would provide that information. Note that the metrics should be easy to read and easy to understand for everyone in the organization. • Analyse the impact of the selected agile practices. If implemented, what is the expected impact on the defect metrics? How do you interpret the data? • Plan how the defect data will be collected. Is there a need to update instructions on how to manage the defects with a defect tracking tool?
Collecting feedback from other sources	• Feedback could include information on what agile practices are used, on commitment to the change, on perception of the change (is the change seen by personnel e.g. as improved software quality), on the motivation level for the change, on the level of knowledge on managing defects in the agile software development, and on the teams' visibility to progress in the transformation. • Plan how the data will be collected.

18 FINAL REMARKS

In this study the primary focus was to identify the impact of agile methods on software development from a defect management point of view in a large organization during the initial phase of adopting agile practices. Additionally it also provided an insight into large-scale adoption of agile methods in general. In order to succeed in this kind of environment, several key issues were identified. First, among the individual agile practices, continuous integration seems to play a critical role. Especially with large systems, where multiple teams are promoting their code changes into the system build, feedback on the changes needs to be provided fast and accurately. Implementing a working environment for continuous integration requires effort and thus this should be one of the first items to be implemented in an organization going towards agile development.

The second key issue to consider is the division of work between the distributed teams. It has been noticed that a co-located team with responsibility on a wide entity is motivated and highly committed to making those features work for which they have responsibility. In large systems there are dependencies between different components of the system, thus the teams cannot be totally independent from the progress in the other teams. Time spent on planning the division of feature responsibilities as bigger entities between the locations will pay back, and very clear communication on the priorities between the features under development supports the work greatly.

Finally, the third key issue to consider is the agile transformation itself. Certain boundaries need to be set from day one, for example, a common sprint schedule and an agreement on what happens when the system level build fails. In addition to this, teams should be encouraged to try out the different agile practices, and to find their own ways of working within agreed, organizational boundaries. An agile coach network could be established to support the teams in their agile transformation, as especially in the beginning it might be difficult to select which practices would be beneficial and should be exercised.

Agile transformation is a journey that will undoubtedly be a challenge to any organization, especially if the organization is large and widely spread over different locations. But it is indeed possible to adopt agile methods in a large company as well, and get benefits. How great the benefits are, and how long and challenging the journey to get those is - that depends on the people in the organization, and on their willingness to change. The motivation to change will be greater if there is a real problem to be solved in the software development organization, and the transformation is not done only because agile is the next popular thing to be tried.

The agile journey will not (or at least should not) stop after the transformation, but improvement and learning continue based on the lessons learned and issues encountered. In future the research could be extended to explore how large organizations evolve after their

first initial years of agile, and how they are able to take the next step from practicing agile methods towards a continuously improving organization. It would be particularly interesting to observe which practices are continued or discontinued after the initial agile startup, and what are the reasons behind that.

References

Abrahamsson, P. (2003): Extreme Programming: First Results from a Controlled case study. In Proc EUROMICRO-SEEA, IEEE, 259 – 266.

Abrahamsson, P., Oza, N., and Siponen, M.T. (2010): Agile software development methods: a comparative review. Agile Software Development , Springer, 31-59.

Agile Manifesto (2001): www.agilemanifesto.org

Auvinen, J., Back, R., Heidenberg, J., Hirkman, P., and Milovanov L. (2005): Improving the engineering process area at Ericsson with Agile practices. A Case Study. TUCS Technical report No 716, University of Turku.

Basili V. (1980): Models and metrics for software management and engineering. Proc. American Society of Mechanical Engineers (ASME) Century International Computer Technology Conference (invited paper).

Basili, V.R. (1992): Software modeling and measurements: yhe Goal/Question/Metric paradigm. ACM Technical Report.

Beck, K. (2000): Extreme Programming Explained: Embrace Change, 2nd. ed., Addison-Wesley.

Beck, K. (2003): Test-Driven Development by Example. Addison Wesley.

Begel, A. and Nagappan, N. (2008): Pair programming: what's in it for me? Proc. the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08), IEEE, 120–128.

Benbasat, I., Goldstein, D.K., and Mead, M. (1987): The case research strategy in studies of information systems. Management Information Systems Research Center, University of Minnesota (MISQ) 11(3), 369–386.

Benefield, G.(2008): Rolling out agile in a large enterprise. Proc. the 1st Annual Hawaii International Conference on System Sciences. IEEE ,462 .

Bipp, T., Lepper, A., and Schmedding, D. (2008): Pair programming in software development teams - an empirical study of its benefits. Information and Software Technology 50(3), 231-240.

Black, R (2004): Critical Testing Processes – Plan, Prepare, Perform, Perfect. Addison Wesley, Boston, USA.

Boehm B. and Basili V.R. (2001): Software defect reduction top 10 list, Computer, 34(1), 135–137.

Boehm, B.W., Brown, J.R., and Lipow, M. (1976): Quantitative evaluation of software quality. Proc. Conference on software engineering, ACM, 592-605.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., and Visaggio, C.A. (2007): Evaluating performances of pair designing in industry. Journal of Systems and Software, 80(8), 1317-1327.

Card D. (2003): Managing software quality with defects, Crosstalk, 4-7.

Catal, C. and Diri, B.(2008): A fault prediction model with limited fault data to improve test process. Proc. 9th international Product Focused Software Development and Process Improvement (Profes), 244-257.

Ceschi, M., Sillitti, A., Succi, G., and Panfilis, S.D. (2005): Project management in plan-based and agile companies. IEEE Software 22(3), 21–27.

Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus,D.S., Ray, B.K., and Wong, M.Y. (1992): Orthogonal defect classification – a concept for in-process measurements, IEEE Transactions on Software Engineering, 18(11), 943-956.

Chow, T. and Cao, D-B.(2008): A Survey study on critical success factors in agile software projects. Journal of Systems and Software 81(6), 961-971.

Cloke, G.(2007): Get your agile freak on! Agile adoption at Yahoo!Music. Proc Agile ’07, IEEE Computer Society, 240–248.

Cockburn, A: and Highsmith, J. (2001): Agile software Development: The People Factor. IEEE Computer 34(11), 131-133.

Cohen, D., Lindvall, M., and Costa, P. (2004): An introduction to agile methods. Advances in computers 62. Elsevier.

Cohn, M. and Ford, D. (2003): Introducing an agile process to an organization. Computer, 36(6), 74-78.

Cohn, M. (2005): Agile Estimating and Planning (1. ed.). Prentice Hall PTR.

Conboy, K., Coyle S., Wang X., and Pikkarainen, M. (2011): People over process: key challenges in agile development. IEEE Software 28(4), 48-57.

Concas, G., DiFrancesco, M., Marchesi, M., Quaresima, R., and Pinna, S. (2008): An agile development process and its assessment using quantitative object-oriented metrics. Proc. 9th international conference on Agile Processes and eXtreme Programming in Software Engineering, 83-93.

Crispin, L. and Gregory, J. (2009): Agile Testing. A Practical Guide for Testers and Agile Teams. Addison-Wesley.

Damm, L.O. and Lundberg, L.(2006): Results from introducing component level test automation and Test-Driven Development. Journal of Systems and Software, 79(7), 1001-1014.

Daskalantonakis, M. K. (1992): A Practical view of software measurement and implementation experiences within Motorola. IEEE Transactions on Software Engineering, 998-1010.

DeClue, T.H. (2003): Pair programming and pair trading: effects on learning and motivation in a CS2 course. Journal of Computing Sciences in Colleges, 18(5).

Duvall, P. M. (2007): Continuous Integration. Improving Software Quality and Reducing Risk. Addison-Wesley.

Fenton, N. E. and Pfleeger, S.L. (1998): Software Metrics: A Rigorous and Practical Approach. 2nd edition, PWS Publishing Company.

Florac, W. (1992): Software quality measurement: A framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22.

Fowler, M. (1999): Refactoring: Improving the Design of Existing Code. Addison Wesley.

Fredericks, M. and Basili, V.(1998): Using defet tracking and analysis to improve software quality. DACS technical report.

Goeschl S., Herp M., and Wais, C. (2010): When agile meets OO testing – a case study. Proc. of the 1st Workshop on Testing Object-Oriented Systems (ETOOS '10).

Gras J-J.(2004): End-to-End Defect Modeling, IEEE Software, 98-100

Grinter, R.E (1995): Using a configuration management tool to coordinate software development. Proc. Conference on organizational computing systems, 168-177.

Hall, T. and Fenton, N. (1997): Implementing effective software metrics programs. IEEE Software 14(2), 55-65.

Hannay, J. and Benestad H. (2010): Perceived productivity threats in large agile development projects. Proc. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'10).

Hartmann, D. and Dymond, R.: Appropriate agile measurement: using metrics and diagnostics to deliver business value. Proc. Agile 2006, 128-134.

Hendrickson, E. (2008): ATDD at http://testobsessed.com/2008/12/08/acceptance-test-driven-development-atdd-an-overview/

Herbsleb, J.D. and Grinter, R.E. (1999): Splitting the organization and integrating the code: Conway's law revisited. Proc. International Conference on Software Engineering, 85-95.

Herbsleb, J.D., Mockus, A., Finholt, T.A., and Grinter, R.E. (2000): Distance, dependencies, and delay in a global collaboration. Proc. ACM Conference on Computer Supported Cooperative Work, 319-328.

Hodgetts, P. (2004): Refactoring the development process: experiences with the incremental adoption of agile practices. Proc. Agile Development Conference. IEEE Computer Society, 106-113.

Huang, L. and Holcombe, M. (2009): Empirical investigation towards the effectiveness of Test First programming. Information and Software Technology, 51(1),182-194.

Huo, M., Verner, J., Zhu, L. and Babar, M.A. (2004): Software quality and agile methods. Proc. Computer software and applications conference, IEEE Computer society, 520-525.

Ileva, S., Ivanov, P., and Stefanova, E. (2004): Analyzes of an agile methodology implementation. Proc. Euromicro Conference, IEEE Computer Society Press, 393-407.

IEEE 1061 (1998): IEEE Standard for a Software Quality Metrics Methodology

ISO 8402 (1986): Quality Vocabulary, in International Organization for Standardization.

ISO/IEC 9126-1 (2001): Software engineering- Product quality- Part 1: Quality model.

ISO/IEC 9126-2 (2002): Software engineering -Product quality- part2: External metrics.

ISO/IEC 9126-3 (2002): Software engineering -Product quality- part3: Internal metrics.

ISO/IEC 25000 (2005): Software product Quality Requirements and Evaluation standard.

Jick, Todd D. (1979): Mixing qualitative and quantitative methods: triangulation in action. Administrative Science Quarterly 24 (4), 602-611.

Jones, C. (1995): Patterns of software systems: failure and success. IEEE Computer Society, 86 – 87.

Johnson J (2002): Keynote speech: build only the features you need. Proc. 4th international conference on extreme programming and agile processes in software engineering (XP 2002).

Johnson, M. (2004): Gallup study reveals workplace disengagement in Thailand. The Gallup Management Journal.

Jäntti, M. (2006): Difficulties in establishing a defect management process: A Case study. Proc. The 6th International Conference on Product Focused Software Development and Process Improvement (PROFES), LNCS 4034, Springer, 142-150.

Kahn, W.A. (1990): Psychological conditions of personal engagement and disengagement at work. Academy of Management Journal, 33(4), 692-724.

Kaner C. and Bond W.P.(2004): Software engineering metrics: what do they measure and how do we know?, Proc. 10th Int. Software Metrics Symposium

Kaner, C., Falk, J. and Nquyen, H.Q.(1999): Testing Computer Software. John Wiley and sons, USA.

Kettunen, P. and Laanti, M. (2007): Combining agile software projects and large-scale organizational agility. Published online 13 July 2007 in Wiley InterScience.

Kerievsky, J. (2004): Refactoring to Patterns. Addison Wesley.

Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S., and Goel, N. (1996): Early quality prediction: a case study in telecommunications. IEEE Software 13(1), 65-71.

Kogure, M. and Akao, Y. (1983): Quality function deployment and CWQC in Japan. Quality Progress, 16(10), 25-29.

Kokkola, M. (2004): Arranging defect management in an agile development process. Seminar in software engineering, Helsinki University of Technology.

Kotter, J. (1995): Leading change: why transformation efforts fail. Harvard Business Review, 73 (2), 59–67.

Kular, S., Gatenby, M., Rees, C., Soane, E., and Truss, K. (2008): Employee engagement: a literature review. Working Paper series 19. Kingston Business School, Kingston University.

Laanti, M. (2010): Agile transformation study at Nokia – one year after. Proc. Lean enterprise software and systems. Springer LNBIP 65, 3-19.

Laanti, M., Salo, O. and Abrahamsson, P.(2011): Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. Information & Software Technology 53(3), 276-290.

Larman, G. and Vodde, B. (2008): Scaling Lean and Agile Development. Thinking and Organizational tools for large-scale scrum. Addison-Wesley.

Larsson, A.(2003): Making sense of collaboration: the challenge of thinking together in global design teams. Proc. International ACM SIGGROUP Conference on Supporting Group Work, ACM Press, 153-160.

Lawrence R. and Yslas, B. (2006): Three-way cultural change: introducing agile with two non-agile companies and a non-agile methodology. Proc. AGILE 2006, IEEE Computer Society, 255-262.

Laymann, L., Williams, L. and Cunningham, L.(2004): Exploring extreme programming in context: an industrial case study. Proc. Agile Development Conference, 32-41.

Lee, S. and Yong, H.S. (2010): Distributed agile: project management in a global environment. Empirical Software Engineering, 15(2), 204-217.

Leffingwell, D. (2007): Scaling Software Agility. Best Practices for Large Enterprises. Addison-Wesley.

Lewin, K. (1946): Action research and minority problems in resolving social conflicts. American Psychological Association, 143-152.

Li, J., Moe, B. and Dybå, T. (2010): Transition from a plan-driven process to Scrum: a longitudinal case study on software quality. Proc. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010).

Lifshitz, G., Kroskin, A. and Dubinsky, Y. (2008): The story of transition to agile sw development. Proc. The 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP 2008), Springer LNBIP, 212-214.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., and Kahkonen, T.(2004): Agile software development in large organizations, IEEE Computer 37(12), 26-34.

Mays, R.G., Jones, C.L., Holloway G.J., and Studinski, D.P. (1990): Experiences with defect prevention, IBM Systems Journal 29(1), 4-32.

MacCormack, A., Kemerer, C.F., Cusumano, M. and Crandall, B. (2003): Trade-offs between productivity and quality in selecting software development practices. IEEE Software 20(5), 78-85.

Mahanti, A.(2006): Challenges in enterprise adoption of agile methods – a survey. Journal of Computing and Information Technology CIT 14(3), 197-206.

Maximilien, E.M., and Williams, L. (2003): Assessing test-driven development at IBM. Proc. the 25th International Conference on Software Engineering, USA. IEEE Computer Society, 564-569.

McBreen, P. (2003): Questioning Extreme Programming. Addison-Wesley.

McConnell, S. (1997): Gauging software readiness with defect tracking. IEEE Software 14(3), 135-136.

McDowell, C., Werner, L., Bullock, H. and Fernald, J. (2002): The effects of pair-programming on performance in an introductory programming course. Proc. the 33rd SIGCSE Technical Symposium on Computer Science Education, 38-42.

Misra, S., Kumar, U., Kumar, V., Grant, G. (2006): The organizational changes required and the challenges involved in adopting agile methodologies in traditional software development organizations. Proc. the 1st International Conference on Digital Information Management. IEEE, 25-28.

Moser R., Abrahamsson P., Pedrycz W., Sillitti A., and Succi G. (2008): A Case study on the impact of refactoring on quality and productivity in an agile team. Springer LNCS 5082/2008, 252-266.

Nagappan, N., Maximilien, E.M., Bhat, T., and Williams L. (2008): Realizing quality improvement through test driven development: results and experiences of four industrial teams. Empirical Software Engineering, 13(3).

Nerur, S. and Balijepally, V. (2007): Theoretical reflections on agile development methodologies. Communications of ACM, 50(3), 72-78.

Nerur S., Cannon A., Balijepally V., and Bond P. (2010): Towards an understanding of the conceptual underpinnings of agile development methodologies. Agile Software Development by Dingsøyr T., Dybå T. and Brede N. (eds.), Springer, 15-29.

Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005): Challenges of migrating to agile methodologies. Communications of the ACM, 48(5).

Niessink, F. and Vliet, H. (2001): Measurement program success factors revisited. Information and Software Technology 43(10), 617-628.

Petersen, K. and Wohlin C. (2010): The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. Springer Empirical Software Engineering 15, 654-693.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. (2008): The impact of agile practices on communication in software development. Empirical Software Engineering 13, 303-337.

Poppendieck, M. and Poppendieck, T. (2007): Lean Software Development. Addison-Wesley.

QAI research report number 8 (1995): Establishing a software defect management process.

Qumer, A. and Henderson-Sellers, B. (2008): A framework to support evaluation, adoption and improvement of agile methods in practice. Science Direct issue 81, 1899-1919.

Qumer, A., Henderson-Sellers, B., and McBride, T. (2007): Agile adoption and improvement model. Proc. of European and Mediterranean Conference on Information Systems, (EMCIS).

Ranganath, P. (2011): Elevating teams from “doing” agile to “being” and “living” agile. Proc Agile 2011 conference, 187-194.

Robson, C. (2002): Real World Research. Blackwell, (2nd edition).

Rohunen, A., Rodriguez, P., Kuvaja, P., Krzanik, L. and Markkula, J. (2010): Approaches to agile adoption in large settings: a comparison of the results from a literature analysis and an industrial inventory. Proc. Product Focused Software Development and Process Improvement (Profes 2010), Springer LNCS, 77-91.

Royce, W. (1970): Managing the development of large software systems. Proc. IEEE WESCON 26, 1–9. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Runeson P. and Höst M. (2009): Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14, 131–164.

Sanchez, C., Williams, L., and Maximilien, E.M. (2007): On the sustained use of a test-driven development practice at IBM. Proc. AGILE 2007, 5-14.

Sandusky R. J. and. Gasser, L. (2005): Negotiation and the coordination of information and activity in distributed software problem management. Proc. ACM SIGGROUP Conference, 187-196.

Schatz, B. and Abdelshafi, I. (2005): Primavera gets agile: a successful transition to agile development. IEEE Software 22, 36-42.

Schwaber, K. and Beedle, M. (2002): Agile Software Development with Scrum. Prentice Hall

Sfetsos P. and Stamelos I. (2010): Empirical studies on quality in agile practices: a systematic literature review. Proc. QUATIC 2010.

Sidky, A. and Arthur, J. (2007): A disciplined approach to adopting agile practices: the agile adoption framework. Innovations in Systems and Software Engineering, Springer, 203-216.

Sidky, A., Arthur, J., and Bohner, S.(2007): Determining the applicability of agile practices to mission and life-critical systems. Proc. 31st Annual IEEE Software Engineering Workshop, IEEE Computer Society, 3-12.

Sillitti, A., Ceschi, M., Russo, B., and Succi, G. (2005): Managing uncertainty in requirements: a survey in documentation-driven and agile companies. Proc. 11th IEEE international symposium on software metrics (METRICS 2005), 10-17.

Soin, S. (1992): Total Quality Control Essentials. McGraw-Hill International editions.

van Solingen, R. and Berghout, E. (1999): The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development. McGraw-Hill International (UK) Limited.

Stephens, M. and Rosenberg, D. (2003): Extreme Programming Refactored: the Case Against XP. Apress, Berkeley.

Stringer, E.T. (1999): Action Research, 2nd edition. Sage Publishing.

Susman, G.I. and Evered, R.D. (1978): An assessment of the scientific merits of action research. Administrative Science Quarterly 23(4).

Svensson, H. and Höst, M. (2005): Introducing an agile process in a software maintenance and evolution organization. Proc. 9th European conference on software maintenance and reengineering (CSMR 2005), 256-264.

Tan, C.H. and Teo H.H. (2007): Training future software developers to acquire agile development skills. ACM Communications 50(12), 97-98.

Tian J.(2004): Quality-Evaluation Models and Measurement. IEEE Software, 21(3), 84-91.

Tseng, Y. and Lin, C. (2011): Enhancing enterprise agility by deploying agile drivers, capabilities and providers. Elsevier Information Sciences 181, 3693-3708.

Vilkki, K.(2009): Impacts of Agile Transformation. Flexi Newsletter 1/2009.

Wils, A, Baelen, SV, Holvoet, T., and Vlaminck, KD. (2006): Agility in the avionics software world. Proc. The 7th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP2006), 123-132.

Yin, R. (1994): Case Study Research: Design and Methods. Sage Publishing.

Yin, R. (2003): Case study research. Design and methods, 3rd ed. Sage Publishing.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-2790-6
ISSN 1459-2045