Developing Support for Agile and Plan-Driven Methods

Harald Svensson

Royal Institute of Technology
Department of Computer and Systems Sciences

December 2005
Consummatum est
Abstract

Software development processes provide means to develop software in a structured and disciplined way. Although these processes provide support for how to develop software, there is still a challenge to successfully apply them in practice. Thus, there is a need to support management of software development processes so that their use generate as good results as possible. The thesis presents support for agile and plan-driven methods, which represent different approaches for developing software.

The support for agile methods was developed after performing a case study and a survey, in order to gather empirical data. The support for plan-driven methods was developed after applying a combination of research methods such as a controlled experiment, grounded theory and theoretical analysis. The developed support for managing agile and plan-driven methods is based on the results from an extensive empirical basis consisting of five studies, with both qualitative and quantitative approaches.

The developed support for agile methods consists of a set of guidelines for organizations to consider when introducing agile methods. The developed support for plan-driven methods concerns support for the Business Process Analyst role to manage its work in a software development context. Further, the Personal Software Process is allocated to several roles in order to increase its applicability.
Acknowledgements

It has been a most rewarding experience of both conducting the research presented in this thesis and writing the actual thesis. During this time I have had the fortune to receive excellent supervision from Dr Martin Höst and Professor Harald Kjellin. I thank both of you for your kind support.

Further, I would like to thank my mother, father and brother for their support and encouragement during my education, both before and after I embarked on my doctoral studies. I also thank Liliann for her support and sound advice to work one year in the industry after receiving my Licentiate of Technology.

During my time as a doctoral student I have met some wonderful people at the Department of Computer and Systems Sciences. A special thank you for all the fun goes to Mats Skoglund, Janis Stirna, Kjell Nåckros, Fredrik Björck, Eriks Sneiders, Nasrin Shakeri and Henrik Bergström.
Contents

I Prologue 15

1 INTRODUCTION 17
  1.1 Overview ........................................ 17
  1.2 Background ...................................... 17
  1.3 Structure of the Thesis .......................... 19

2 RELATED WORK 21
  2.1 Overview ........................................ 21
  2.2 Software Process Improvement .................... 21
    2.2.1 Process Analysis and Modelling ............ 24
    2.2.2 Introducing Software Development Processes .... 24
  2.3 Plan-Driven Methods ................................ 26
    2.3.1 Plan-Driven Methods Overview ............... 26
    2.3.2 Personal Software Process .................... 27
  2.4 Agile Methods .................................... 31
    2.4.1 Agile Methods Overview ..................... 32
    2.4.2 Extreme Programming ......................... 33

3 RESEARCH MOTIVATION 37
  3.1 Overview ........................................ 37
  3.2 Research Problem ................................ 37
  3.3 The Studies’ Relation to the Research Problem .... 39
  3.4 The Studies’ Problems and Questions ............ 42

4 RESEARCH METHODOLOGY 45
  4.1 Overview ........................................ 45
    4.1.1 Research Strategies ......................... 45
    4.1.2 Research Methods ............................ 47
  4.2 Description of the Research ...................... 48
    4.2.1 Overview .................................... 48
    4.2.2 Study 1: Code Review Efficiency ............ 49
    4.2.3 Study 2: SPI Support Based on Non-Technical Criteria 50
    4.2.4 Study 3: Individual SPI Support in RUP ....... 51
CONTENTS

II Support for Plan-Driven Methods

5 STUDY 1: CODE REVIEW EFFICIENCY

5.1 Background and Problem
5.2 Related Work
5.3 Design of the Study
5.4 Yield Management
  5.4.1 Analysis
  5.4.2 Summary
5.5 Economics of Defect Removal
5.6 Concluding Remarks

6 STUDY 2: SPI SUPPORT BASED ON NON-TECHNICAL CRITERIA

6.1 Background and Problem
6.2 Related Work
6.3 Business Process Analyst
6.4 Research Methodology
  6.4.1 Selection of Study Participants
  6.4.2 Developing the SPI Support
  6.4.3 Validity
6.5 Framework Overview
6.6 Results
  6.6.1 Assessment Results
  6.6.2 Interview Results
6.7 Concluding Remarks

7 STUDY 3: INDIVIDUAL SPI SUPPORT IN RUP

7.1 Background and Problem
7.2 Related Work
7.3 Rational Unified Process
7.4 Comparing RUP and PSP
  7.4.1 Application of the Processes
  7.4.2 Terminology of the Processes
7.5 Incorporating PSP into RUP
7.6 Concluding Remarks
III  Support for Agile Methods  109

8  INTRODUCTION TO STUDY 4  111
   8.1 Overview ........................................... 111
   8.2 Background .......................................... 111
   8.3 Study Context ...................................... 112
      8.3.1 The Company .................................. 112
      8.3.2 Study Participants ............................. 113
      8.3.3 Development Environment ...................... 113
   8.4 Introducing the Agile Method ....................... 114
      8.4.1 Original Process ................................ 116
      8.4.2 Introduced Changes ............................. 118
      8.4.3 Comparing the Introduced Process to XP ....... 118

9  STUDY 4.1: QUANTITATIVE MEASURES ON AGILE DEVELOPMENT  121
   9.1 Background and Problem ............................. 121
   9.2 Related Works ...................................... 122
   9.3 Research Methodology .............................. 123
      9.3.1 Design ......................................... 124
      9.3.2 Data management ................................ 124
      9.3.3 Selection of Study Participants ................ 126
      9.3.4 Selection of Metrics ............................ 126
      9.3.5 Validity ....................................... 127
   9.4 Results ............................................. 127
   9.5 Interpreting the results ............................ 131
   9.6 Main Conclusions ................................... 134
   9.7 Concluding Remarks ................................ 135

10 STUDY 4.2: QUALITATIVE MEASURES ON AGILE DEVELOPMENT  137
   10.1 Background and Problem ............................ 137
   10.2 Related Works ..................................... 138
   10.3 Research Methodology ............................. 139
      10.3.1 Selection of Study Participants ............... 140
      10.3.2 Qualitative analysis ........................... 141
      10.3.3 Subjective Rankings ............................ 144
      10.3.4 Validity ....................................... 145
   10.4 Results ............................................ 146
      10.4.1 Results from Interviews ....................... 146
      10.4.2 Results from Observations ..................... 148
      10.4.3 Results from Subjective Rankings ............. 148
   10.5 Main Conclusions .................................. 150
   10.6 Concluding Remarks ................................ 152
14.3.2 Study 2: SPI Support Based on Non-Technical Criteria
14.3.3 Study 3: Individual SPI Support in RUP
14.3.4 Study 4.1: Quantitative Measures on Agile Development
14.3.5 Study 4.2: Qualitative Measures on Agile Development
14.3.6 Study 4.3: Agile Collaboration Between Team and Organization
14.3.7 Study 5: A Survey on Agile Development
14.4 Analysis of Validity
14.5 Future Work

A TERMS AND CONCEPTS USED IN SPI
# List of Figures

2.1 The main activities in SPI ..... 23  
2.2 The PSP levels ..... 28  
3.1 Relationship between support and process ..... 38  
5.1 LOC reviewed per hour versus code review yield ..... 65  
5.2 Average yield for PSP exercises ..... 66  
5.3 Time spent in defect removal activities for Exercises 7-10, with versus without reviews ..... 67  
6.1 The structure of the BPA framework ..... 80  
6.2 The assessment results ..... 81  
7.1 Percentages of RUP workers involved in PSP activities ..... 100  
7.2 PSP disseminated to RUP workers ..... 100  
7.3 Mapping PSP phases to RUP workers ..... 101  
7.4 PSP applied by RUP workers ..... 102  
7.5 Allocation of PSP metrics to RUP workers ..... 103  
9.1 Measurement trend for the metric *Project Velocity* ..... 129  
9.2 Measurement trend for the metric *Load Factor* ..... 129  
9.3 Percentage of completed tasks originating from planning games ..... 130  
9.4 Average time spent with PI’s ..... 131  
9.5 Measurement results for the metric *Estimation Error* ..... 132  
10.1 A general model of qualitative studies ..... 140  
10.2 Rating results for practices concerning positive effect on development ..... 149  
10.3 Rating results for practices concerning ease of introduction ..... 150  
10.4 Measurement trend for the metric *Project Velocity* ..... 152  
11.1 A general model of qualitative studies ..... 159  
12.1 Experience of the typical software engineer ..... 178  
12.2 Development contexts where XP was applied in ..... 179
12.3 The teams’ opinions of XP before it was introduced. . . . . . 180
12.4 Initiator for agile development. . . . . . . . . . . . . . . . . . . . . 181
List of Tables

2.1 A summary of XP’s practices. .............................................. 34
4.1 Strategies, methods and data collection techniques used in
the research. ................................................................. 49
5.1 Code review results. ......................................................... 63
6.1 Structure of the BPA framework. ......................................... 77
7.1 Comparing PSP and RUP. ................................................... 90
7.2 A comparison between PSP and RUP regarding iteration as-
assessment. ................................................................. 91
7.3 A comparison between PSP and RUP regarding process tool
support. ................................................................. 92
7.4 A comparison between PSP and RUP regarding collection
and logging of process data. ................................................. 93
7.5 A comparison between PSP and RUP regarding number of
involved people in the software development process. ............... 93
7.6 A comparison between PSP and RUP regarding process ma-
urity. ................................................................. 94
7.7 Guidelines for introducing PSP into RUP. .............................. 106
7.8 Areas in RUP where PSP may provide support. .................... 107
8.1 Process changes due to introduction of the agile method. ........ 119
8.2 Differences regarding the agile method versus XP. ................ 120
9.1 Metrics used in the study. .................................................. 126
10.1 The team’s codes for experiences of implementing the practice
Continuous integration. .................................................. 144
11.1 Experiences made by the organization when collaborating
with the team. ............................................................... 161
11.2 Trends regarding collaboration issues between the team and
its surrounding organization. ............................................. 165
12.1 Summary of survey questions. ....................... 176
12.2 Reasons for introducing XP. .......................... 181
12.3 Quality characteristics for developed products. .... 182
Part I

Prologue
Chapter 1

INTRODUCTION

1.1 Overview

The purpose of this chapter is to provide a background and motivation of the thesis’ research problem. Further, the research papers which this thesis is based on are presented followed by an overview of the structure of the thesis.

1.2 Background

Since the industrial revolution emerged in the early nineteenth century, the manufacture industry has been a main contributor to the welfare of the society. As the demands increased on productivity and quality, it became important to investigate ways to improve the development process. By measuring and analyzing the way products were developed opportunities for improvements could be identified, which resulted in more efficient development processes. Today, software industry has a major impact on society’s welfare. The software industry has grown dramatically and affects many parts of the modern society. Software products are used in most business sectors and for many reasons. As for the manufacturing industry, it is important for the software industry to also find ways to improve the development process. The global cost for software development is huge. Projects are seldom finished in time, and it is not uncommon that software is released with faults. A survey conducted by the Standish group [50] investigated the overall rate of failure for software development projects. The survey analyzed over 13 000 projects and only about one third of these projects were successfully completed. The rest of the projects did not either deliver the right functionality, exceeded the budgets or did not deliver the expected software at the agreed date. This may be a result of fierce competition in the software development community, but also due to the complexity involved in software engineering.

Thus, it is important for the software community to continuously inves-
CHAPTER 1. INTRODUCTION

Examine means for improving software development, as mentioned for example by Sommerville [92]. One approach is to apply formally defined software development processes as they provide an order and structure to the complex activity of developing software. However, although a process\textsuperscript{1} is based on a sound theoretical basis it may not be a trivial task to apply it efficiently in practice. For example, there are numerous aspects to consider when introducing a process into an organization, or realizing how to adapt a process to its software development context. Thus, it is important to support management of software development processes as the user of the process (i.e. individual, team or organization) then has the means to apply the process more efficiently. For example, support can either be predefined guidelines how to introduce a process efficiently into an organization. Support can also be in form of metrics and templates which are used when collecting and analyzing process data. Both kinds of support help manage and improve process usage.

Although some processes provide support for process usage, most processes do not provide such support. The main reason for this may be that the support should be based on results from empirical studies which may be difficult to conduct, in order for the support to be valid. Hence, there exists a limited amount of support based on empirical data for software development processes. This aspect has a negative impact on the management of software development processes.

The thesis addresses the problem mentioned above by developing support for managing processes. Support is developed for agile and plan-driven methods which are terms that characterize two different software development approaches. Agile methods are useful when managing unexpected changes due to their informal approach to software development. Plan-driven methods on the other hand provide a disciplined way to develop software which may be useful in a chaotic environment which often characterizes software development contexts. Both approaches are useful as they address important aspects to consider when developing software. The agile method extreme programming, XP [10] was used when developing support for agile methods, and the plan-driven method the Personal Software Process, PSP [42] was used when developing support for plan-driven methods. The motivation for using these processes is presented in Section 3.3. This monography thesis is based on the seven papers presented below.


\textsuperscript{1}The term process is synonymous with the term software development process. Both terms are used as appropriate throughout the thesis.
1.3. STRUCTURE OF THE THESIS


1.3 Structure of the Thesis

The structure of the thesis is as follows. Part 1 provides a basic understanding of the purpose and area of research presented in the thesis. This is followed by an overview of related work. Further, the research problem is presented which is followed by the studies’ subproblems and their associated research questions. The part ends with motivations for the choices of research strategies and methods that were applied in the research. Part 2 includes the research performed on developing support for plan-driven methods. Part 3 presents results of developing support for agile methods. Part 4 is the epilogue of the thesis, which identifies the contribution of the research presented in the thesis. Further, this part discusses validity threats to the
research and identifies future research opportunities. In the discipline of software engineering, there does not exist a homogenous definition of terms in the field. Thus, due to the inherent ambiguity in the semantics of software engineering terms, definitions of relevant terms used in the thesis are provided in Appendix 1.

Process improvement for plan-driven methods is addressed in Part 2 of the thesis, and the improvement of agile development is addressed in Part 3 of the thesis. It should be noted that software engineering is an interdisciplinary research area. The thesis covers research areas such as psychology for example in the form of team building. Further, how humans react to changes probably have an impact on the research presented in the thesis. However, it is not possible to address or include all of these aspects. Therefore the scope of the thesis is limited to software process improvement, agile and plan-driven methods which are believed to be most relevant for the research presented in the thesis.
Chapter 2

RELATED WORK

2.1 Overview

The work presented in the thesis encompasses support for improving the use of software development processes. The work focuses on developing process improvement support at an individual level in software development for plan-driven methods, and providing guidelines for software development organizations how to introduce agile methods. Section 2.2 presents an overview of the Software Process Improvement, SPI, area. The first part of the research presented in the thesis concerns software process improvement for plan-driven methods. Section 2.3 includes a brief presentation on what characterizes plan-driven methods, and an overview of some prominent plan-driven methods is provided in Section 2.3.1. The research on plan-driven methods is based on the Personal Software Process [41], PSP. PSP is used in studies 1 and 3, and to some extent in study 2. Thus, Section 2.3.2 provides a detailed presentation of PSP. The second part of the research concerns support for agile methods. An summary of agile development in general is provided in Section 2.4. Further, an overview of some prominent agile methods is provided in Section 2.4.1. The research on agile methods is based on extreme programming [10], XP. Thus, Section 2.4.2 provides a detailed presentation of XP.

2.2 Software Process Improvement

This section presents a summary of the area Software Process Improvement, SPI. This is followed by an overview of the more specific research areas within SPI that the thesis addresses, which is Process Analysis and Modelling and Introducing Software Development Processes. The research in the thesis that addresses support for plan-driven methods, includes modelling and analysis of software development processes. Thus, the section includes an overview of this area. Finally, a summary is provided of the area of introducing soft-
ware development processes into organizations, which was addressed when developing support for agile methods.

Developing software is a complex activity. Software engineers need to learn programming languages and tools to develop software products. However, there is more to software development than just tool usage. The creativity and collaboration effort between developers to a large extent decide the quality of the software product. The software development industry has recognized the importance of applying a defined and mature software development process in order to successfully deliver a software product with expected quality, for instance mentioned by Nedstam [68] and Derniame et al. [25]. As stated by Derniame et al. [25] the research field of software processes has grown considerably the last two decades, which has had an impact on the software development community. Today, the notion of a software process is well-understood and appreciated which is reflected in the industry by job titles such as software process engineer.

SPI is a research area within software engineering that investigates how to assess, support and improve the software development process. In its essence, SPI involves identifying an opportunity for improvement, introducing a change of some sort and assessing whether the expected improvement occurred or not. Thus, an SPI activity consists of the following main steps. The first step involves recognizing an opportunity for improvement and planning the change. An improvement opportunity is identified by examining and analyzing the software process. The analysis may be facilitated by modelling the process which provides a better understanding of the process’ dynamic behavior. The second step refers to implementing the planned changes. Preferably, the introduced changes are tested on a small scale first to minimize disruption on ordinary routines. The purpose of the third step is to determine whether the introduced changes actually achieved the expected improvement or not. This decision is based on analyzing the results of the introduced changes. After evaluating the effects from the introduced changes, a course of action is taken. If the introduced changes had a positive impact, they may be introduced as part of the routine activity. If they were not successful, a new iteration of the improvement cycle is started. A more comprehensive view, provided by Sommerville [92], of SPI activities is presented in Figure 2.1.

Successful SPI initiatives are often the combined use of several strategies. Thus, the area of SPI consists of a number of research areas. As stated by Demarco [24] “you cannot control what you cannot measure”, it is vital to collect process data to be able to assess whether improvements have been gained. This view is further supported for instance by Roberts [83]. An area related to SPI that addresses this need is software measurement. The area of software measurement is concerned with measuring specific attributes of a software product or software development process. By analyzing collected data, an understanding of the current status is gained, whether it is the
2.2. SOFTWARE PROCESS IMPROVEMENT

Before embarking on an SPI initiative, a natural first step is to assess the current ability of an organization to deliver software. This ability is referred to as software process maturity. Thus, the field of software process assessment evaluates organizations’ processes and their capabilities of delivering software. There exist a number of books on the area, for example the ones by Emam et al. [27] and Hunter et al. [45] which provide comprehensive overviews on assessment methods and results of applying them in the industry. The Capability Maturity Model, CMM-SW [46] was developed as a response of the need to have a structured approach for assessing organizations’ software development capabilities. The CMM-SW helps organizations to evaluate their software development capability on a five graded scale. Each maturity level encompasses a number of goals and practices. By fulfilling the goals of a maturity level, an organization is recognized as being capable of developing software of a certain maturity associated with that level. There exist several CMM approaches, and as the software community has realized that many organizations need more than one of these approaches, the Capability Maturity Model Integrated, CMMI [20] was developed which provides a combination of two up to four different types of CMMs. The advantage of using the CMMI approach, compared to the former CMM approach, is that several issues originating from different CMMs (e.g. management of employees and software development) are addressed in an uniformed way when applying the CMMIs. Thus, an organization can address several important issues in the same SPI effort whereas before it would have taken one effort for each separate issue.

The SPICE model [51] is the result of efforts to develop a standard to apply when performing software process assessments. SPICE is an abbreviation and is short for Software Process Improvement and Capability dEtermination. The standard, also referred to as ISO 15504, helps organizations assess software development processes. Like the CMM-SW, it uses maturity levels to classify an organization’s software development capabili-
ity. However, unlike the CMM-SW, it does not require or recommend any specific assessment model but provides a generic framework which guides the assessment procedure.

The large number of institutes and companies involved in software process improvement indicate that it is not a straightforward task to successfully implement SPI initiatives. This is further supported by viewpoints expressed by Gray and Smith [37] which address difficulties associated with software process improvement. One important obstacle seems to be the lack of immediate success. It takes time and effort to implement SPI programs. Opportunities for improvements must be identified, changes planned and introduced and data collected and analyzed before any benefits may be identified. Having this in mind and recognizing the fact that many organizations have a rather short-term view, may explain the difficulties in the industry to successfully introduce and sustain SPI initiatives.

2.2.1 Process Analysis and Modelling

When planning and developing support for plan-driven methods, software development processes were modelled and analyzed to understand how to implement the support. Software process modelling, for instance described by Finkelstein et al. [31], is an area within SPI that models the dynamic behavior expressed by software processes. When modelling a process the key characteristics of the process are illuminated, thus improving the ability to control and monitor the process. There exist different modelling techniques such as finite state machines described by Nuseibeh et al. [70], where a process’ behavior is modelled with focus on transitions between different states. This technique is useful when identifying what causes a transition from one state to another. Another modelling technique is the entity process model for instance described by Carlis et al. [18]. This technique is useful when modelling the entities in a process and their interrelations. An entity can describe various objects or artifacts such as a person, but it can also for example be a document or the source code for an application. Essentially, these two approaches (i.e. finite state machines and entity process modelling) model the data structure and flow inherent in software development processes. Both these approaches were applied when developing support for plan-driven methods.

2.2.2 Introducing Software Development Processes

The issue of introducing software development processes into organizations has many aspects in common with the area of change management, which has received a substantial amount of research such as the works conducted by Kotter [56] for instance. However, more specific research on introducing software development processes has not been conducted to a sufficient level
2.2. SOFTWARE PROCESS IMPROVEMENT

which could be concluded after performing a literature survey on the issue. An organization's improved understanding of how to introduce and adapt processes in different contexts would help to improve its overall software development effort as inappropriate measures can be avoided. This viewpoint is shared for instance by Ahnve [7] and Abrahamsson et al. [6].

The introduction of software development processes is identified as an important issue to consider in software development both by the academia and the industry. For instance, many consulting firms in the SPI area offer assistance for introduction and adaptation of software development processes. The concept of introducing software development processes may have different meanings for different people. Thus, it is motivated to discuss what the concept means in the thesis. The activity of introducing a software development process is defined in the thesis to be composed of three parts. These parts have been derived after reading in our opinion relevant work on the issue (e.g. [56], [60] and [7]). The first part involves preparation activities before the process is actually introduced. During this phase it is important to define what the purpose of introducing the process is, to investigate the organization's attitude towards the SPI initiative and to define a course of action how to introduce the process as mentioned by Ahnve [7]. It should also be noted that it takes time to perform an SPI program, partly due to the need to collect and analyze data in order to assess whether improvements have been gained or not. Another aspect mentioned by Karlström [54], is the typical human response to change which is classified to involve phases such as denial, anger, depression and finally acceptance. Considering this, although stereotyped, view of the different emotional stages encountered when dealing with change, it is understandable that it takes time to complete. This fact should be included in the planning phase. Further, to overcome the resistance to change it is important to create a sense of awareness of the necessity to introduce the change, as stated by Kotter [56]. When introducing change into an organization, as is the case when introducing a software development process, there is seldom an unanimous decision advocating this change. Thus, it is important to create a coalition of people high up in the organizational hierarchy with a substantial amount of political power, that drives and supports this change. Otherwise, it is not unlikely that eventually people will stop using the introduced process as argued by Kotter [56].

The second part of the introduction process addresses the selection of the software development process. It is the opinion of the author of this thesis, further support by Ahnve [7], that these decisions are often made on a more or less uninformed basis. Organizations tend to select those processes that are used by others, instead of taking a decision based on how well a process works in a certain development context. When selecting a process, usage requirements on the process should be addressed such as if the process should be applied at a team or organizational level, or if
the process should be formal or informal to its nature. After identifying a possible process candidate it is wise to conduct a pilot study on a small scale where the process is introduced to a team, instead of being introduced to the whole organization. By analyzing the experiences from the pilot study, an informed decision can be made whether and how to introduce the software development process to the rest of the organization. This approach was used in the research presented in the thesis.

The last part of the introduction process concerns the efforts needed to successfully apply a software development process on a continuous basis. Important aspects to address are educating the organization on the process, process adaptation issues and the need to provide people with feedback on the results of using the process as stated by Lientz and Rea [60]. Both qualitative and quantitative data are useful to analyze, since qualitative data can provide information on how the organization perceives the usage of the process, and quantitative data may present an overview of process performance.

2.3 Plan-Driven Methods

The purpose of this section is to characterize the meaning of plan-driven methods, and to present some prominent plan-driven methods. The section is ended with a more detailed presentation of the plan-driven method PSP, which was used in Study 1, 2 and 3 in the thesis.

Plan-driven methods are characterized by having well-defined development phases offering opportunities for continuous improvement, as stated by Boehm and Turner [13]. As the term implies, there is a planning aspect inherent in these methods. Thus, the methods contain detailed descriptions of activities, work flows and roles and so forth in order to facilitate the planning activity. The development activity consists of following a series of carefully specified phases, where the output from one phase is the input to the next phase. Further, there is a focus on thorough documentation in each phase. The level of detail and content in these kinds of processes require training of people in their application to apply them efficiently. However, an advantage with plan-driven methods is the repeatability of the process due to the level of detail of its contents.

2.3.1 Plan-Driven Methods Overview

There exist several plan-driven methods. Below, a summary is provided of some prominent ones. However, PSP is not included in the summary as it is presented in greater detail in Section 2.3.2. The intention is not to provide an exhaustive description of all plan-driven methods, but to identify and highlight important contributions.
2.3. PLAN-DRIVEN METHODS

**CMMI.** CMMI is an abbreviation of Capability Maturity Model Integration. The CMMI are not methods in the sense that they are applied by individuals or teams. These models state, at an organizational level, what should be addressed in order to increase an organization’s software development capability. The two examples below provide means *how* to implement what the CMMI models identify, at a team and individual level. The CMMI models combine best practices from different development areas such as systems engineering and process development, to be applied in an unified effort. Thus, the advantage compared to former CMM models is that different software development areas (e.g. systems engineering and supplier sourcing) are included in one model, instead of addressing one area in one model as is the case with the CMMs. The CMMI models can, like the former CMM models, be used as appraisal models when assessing an organization’s software development capability.

**Team Software Process.** The Team Software Process [43], TSP, is a plan-driven method at a team level based on the PSP which is a plan-driven method at an individual level. TSP was developed to provide an environment which facilitated the use of PSP. Thus, the team members that apply the TSP are trained PSP engineers. In addition of applying the contents in PSP, TSP also has a number of roles defined in order to synchronize the team’s development efforts. Both PSP and TSP provide instructions how to implement the contents in CMM-SW. TSP provides support *how* to implement the CMM-SW’s highest maturity level in a team development environment, whereas the CMM-SW states *what* to address in software development at an organizational level. The TSP also includes support for identifying and assessing risks that may occur in a software development project.

**2.3.2 Personal Software Process**

Most software development processes provide support for software development at a team or organizational level. The number of software development processes that provide support at an individual level are few. However, the Personal Software Process [41], PSP is a plan-driven method at an individual level that provides support for the software engineer to improve his/her software development process. PSP has seven different maturity levels. The first level, PSP0, is the most basic whereas the last level, PSP3, is the most mature software development process. The software engineer gradually applies higher levels of PSP, as the software development process evolves. PSP provides metrics, process-steps, and templates to the software engineer. By applying metrics and templates and following PSP’s process steps, the engineer can collect process data. By analyzing these data, he/she can focus on
CHAPTER 2. RELATED WORK

weak aspects in the software development process and find ways to improve them.

The concept of a project in PSP is not the whole development project undertaken by a company, where several software engineers develop software. Rather, a project in PSP sense is an individual undertaking, e.g. one or several modules to be developed, or a subsystem, depending on its size. Humphrey [41] recommends that a PSP project should concern development of a piece of software with the size 200-300 Lines Of Code (LOC). Of course, this figure may vary depending on how a LOC is defined. However, the number of LOC gives a hint of the recommended size of a PSP project. The development work is broken down into the phases Planning, Development, and Postmortem. The Development phase is further broken down to Design, Code, Compile, and Test. PSP also provides support for how to manage development of large systems consisting of several thousands of LOC. When embarking on development of such large systems, it is recommended by PSP to initially divide the overall design into smaller designs, which are implemented sequentially. In each of these separate designs, a “normal” PSP cycle consisting of design, code, compile and test phases is applied.

The PSP framework has seven levels: PSP0, PSP0.1, PSP1, PSP1.1, PSP2, PSP2.1, and PSP3. Figure 2.2 presents the PSP levels. Each new level is an extension of the software development process. PSP0, PSP1, and PSP2 introduce a minor further step denoted by the suffix .1, e.g. PSP0.1. Each level introduces a set of issues, new or modified from the level below. Process data is recorded on a number of forms, each level introducing more data to record, and more analysis to be made.

![Figure 2.2: The PSP levels.](image-url)
PSP0 The basic idea of PSP0 is to set the baseline, towards which improvement efforts can be compared. At the time of planning an individual project, only the total effort for the project is estimated. Time and defects will be recorded per development phase. When a project is finished, there are data about how much effort was spent in Planning, Design, Code, Compile, Test, and Postmortem, respectively. The defects injected and the defects removed are recorded by defect type and phase. The prerequisite is a defect type standard. Defect recording together with the defect type standard provide valuable information for improvement. For instance, if many defects of a particular type seem to be injected in the design phase a focus may be put on design and alternative design methods considered, or extra design reviews, to mention a few examples.

PSP0.1 PSP0.1 introduces a coding standard to facilitate readability and homogeneity of the code. The Process Improvement Proposal (PIP) is a form for recording the software engineer’s ideas for further improvement of his/her process. The intention is that before the next project is undertaken, the PIP is analyzed and the ideas are realized in the new project if possible. PSP0.1 includes size measurement, and to some extent, size estimation of the code. Not only the total LOC is measured, as majority of the projects do not only involve development of entirely new code. Modifying and adding existing code, and reusing previously developed code may also occur. The size of new and changed code is also estimated in this level.

PSP1 PSP1 introduces the planning element to the software process. Based on time and size measurement data collected from earlier projects, estimations of similar projects can be done. It is important to emphasize that unless a project is similar enough to earlier projects, the measurement data may not provide an adequate basis for estimation. In addition to the planning activity, PSP1 also introduces the test report concept. The Test Report should include input and output data, constraints and purpose of a test, in order to facilitate tests on the code after a future change to ensure that unchanged parts of the code run correctly.

PSP1.1 PSP1.1 introduces task planning by providing a form to record tasks in a project, and estimations for the effort of the tasks in terms of the Earned-Value method, described by Humphrey [42]. Applying the Earned-Value method means that each task is allocated a percentage value where the value depends on how big part the task in question constitutes of the total effort, that is 100%. The Schedule Planning form records the estimated and actual hours expended by calendar time period, applying also the Earned-Value method.
PSP2 PSP2 focuses on early removal of injected defects, raising the quality of the software product. Early removal of defects has a positive economical impact, as defect correction costs increase significantly the later in the development phase the correction is made. The main vehicle for early defect detection are design and code reviews. A defect checklist is constructed, based on data from compile and test phases, to aid in reviews. The checklists are not static, but should be continuously updated to mirror the most frequent and most serious defects. To determine the distribution of the defect types, a Pareto diagram can be constructed.

PSP2.1 PSP2.1 addresses the design process. PSP does not prescribe how to do design, but provides guidelines for determining when a design can be considered complete, in terms of design completeness criteria. Design templates are introduced to ensure that no important aspects in the design are forgotten, and the Design Review Checklist is introduced to aid design reviews.

PSP3 PSP3 presents a framework for large-scale projects developing software products with several thousands of LOC. The paradigm is Divide and Conquer, dividing a large project into a number of small ones and applying PSP2.1 in those. PSP3 introduces Issue Tracking Log (ITL), which is simply a notebook for recording things to remember, like changing a variable name later. Another new element in PSP3 is the Cycle Summary, where process data from previous iterative cycles in a project are recorded.

PSP is not a silver bullet with answers to all software engineering problems. The software engineer must want to be better. As mentioned by Humphrey [42], there are no shortcuts involved in software process improvement. The software engineer must collect and analyze his/her process data, suggest changes and evaluate whether these changes improved the software development process. PSP is intended to be used by software engineers as a means for improving their software development processes. However, although it has been nearly ten years since PSP was introduced it is not used to a large extent by the software development community. Our experience tells us that PSP is viewed as a heavy and complex process to use. This view is based on the fact that PSP focuses on collecting and analyzing process data. Further, this activity takes time and effort which may not be available in software development contexts as they often are characterized by severe time pressure.
2.4 Agile Methods

The purpose of this section is to characterize the meaning of agile development, and to present some prominent agile methods to provide an orientation of the agile development community. The section is ended with a more detailed presentation of the agile method XP, which was used in Study 4 and 5 in the thesis.

In the waterfall model developed by Royce [85], requirements are defined and approved before development starts. Further, a new software development phase is not begun before the former phase is officially ended. There is a focus on documentation, as results and decisions are stored in documents for later analysis. This approach has its advantages, for instance providing project managers with a high level of control in software projects. However, as new prerequisites emerged for software development it became obvious that requirements more often were changed during software development. Further, in many cases customers did not always know at a detailed level what they wanted developed. This had the effect that it became more important to continuously review the software while developing it. These concerns resulted in the introduction of agile development, where the term agile is characterized by Abrahamsson et al. [6] as being incremental, cooperative, straightforward and adaptive. Software is delivered in small releases with short cycles, where developer and customer work together in close cooperation. The method used is easy to understand and modify, and it is adaptable in the sense that last moment changes are facilitated and managed. Agile development is characterized by having an incremental software development approach, based on short iterations. Further, requirements are not determined in advance and may be changed as deeper insight of what to develop is gained.

Agile methods have some fundamental aspects in common regarding how to develop software, compared to the more formal software development approaches. These aspects were formulated in the so called agile manifesto, which was written when some prominent people from the agile community met to discuss the core characteristics of agile development. The agile manifesto, described in [34], states four main aspects that constitute the meaning of agile development:

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

These aspects can be addressed in different ways as seen in the content and structure of agile methods. Scrum is an agile method for instance
described by Abrahamsson et al. [6] which has a stronger focus on how to manage a software development project, compared to XP which focuses more on engineering practices. It should be noted that agile development is not an universally cure to improve software development as a whole. While agile methods address simplicity and speed, they are not suitable in all contexts. Thus, to determine their software development capability more empirical studies on use of agile methods in various contexts should be conducted.

2.4.1 Agile Methods Overview

There exists a number of agile methods. Below, a summary is provided of prominent agile methods described by Abrahamsson et al. [6]. However, although extreme programming is maybe the most well known agile method, it is not included in the summary as it is presented in greater detail in Section 2.4.2. The intention is not to provide a complete picture of the agile development arena but to identify and highlight important contributions.

Scrum. Scrum is a set of practices and roles which helps manage a software development project. The method does not suggest any specific development practices to be used, instead it provides a generic management framework that helps steer software development activities. Scrum consists of three phases. First is the pregame phase where requirements are collected and prioritized, and the software architecture is established. This is followed by the development phase, where software development is overviewed by identifying and controlling environmental and technical variables. Software is developed in iterative cycles, referred to as sprints. Last, the software release which consists of integration, test and documentation activities is addressed in the postgame phase.

The Crystal Family. The crystal family is a set of methods that provides guidelines of using tools, standards and roles in software development. These methods are general in the sense that they do not explicitly state how to perform an activity for example, they merely state what should be performed. Each method is used separately and is characterized by a unique color. The darker the color, the more strict and formal the method. Darker colors are suggested when a project is large or addresses issues such as development of life critical software. The methods in the crystal family address software development from small to large projects.

Feature Driven Development. Feature Driven Development, or FDD in short, applies iterative development which consists of five phases where a focus is put on the design and implementation of software. Thus, maintenance and installation issues for instance are not addressed by
2.4. AGILE METHODS

FDD. After reviewing requirements a feature list is created. Out of this list a number of features are chosen to be implemented during the coming iterative cycle. Further, FDD has a number of roles and general practices to be applied in the development process.

Dynamic Systems Development Method. Dynamic Systems Development Method, or DSDM in short, is based on the idea of not overloading a software development team with work. Instead of fixing the functionality to be developed, and thus adjusting the time and resources needed, time and resources are fixed and the expected functionality will have to be adjusted instead. DSDM consists of five phases, where the two first are only applied once. The first phase reviews the type of project being planned and investigates whether DSDM is suitable to apply. The second phase addresses which business processes to be used in order to properly characterize the software. The last three phases concern software development in iterative cycles. These phases involve the planning and prioritization of work tasks in coming iterations. The last phase helps transfer the system into its production environment.

Adaptive Software Development. Adaptive Software Development, or ASD in short, focuses on developing complex and large systems. ASD consists of three phases where an emphasize is put on the postmortem learning activity, where lessons are learned from previous experiences in earlier development cycles. ASD is general and does not provide any roles or detailed information on activities. ASD only contain three activities: iterative development, feature-based planning and customer focus group reviews, and they do only offer advice what could be done instead of what should be done.

As with all agile methods, there is a lack of empirical studies on applying them in various contexts as stated for instance by Abrahamsson et al. [6]. Thus, it is difficult to say in which contexts they are suitable to apply. Although agile methods seem to fill a gap of emerging software development needs more empirical data is still needed to clarify this issue, as mentioned for instance by Abrahamsson et al. [6] and Reifer [81].

2.4.2 Extreme Programming

Extreme Programming [10], or XP which it is informally called, is the most well known of the agile methods. XP consists of a set of engineering practices for a team to apply when developing software. These practices are based on commonly accepted software development approaches, but what makes XP extreme in some sense is that these approaches are applied at an extreme level. For instance, a system developed by use of XP may be tested,
integrated and built several times a day, instead of maybe once per week or month as is usual otherwise. In addition to these practices, XP consists of roles and values. The purpose of the values is to help the team approach software development with, according to XP, the right attitude.

XP is a process that applies iterative development with intense involvement from the customer. The customer defines so called stories, which describe functionality to be implemented. As soon as a story is implemented, tested and integrated into the system the work of another story begins. Further, the customer decides scope and timing of software releases based on estimates provided by the software developers. The practices help the team develop software in several ways. Table 2.1 presents XP’s practices.

<table>
<thead>
<tr>
<th>Practice</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning game</td>
<td>The team estimates the efforts needed to implement the work planned in the coming iteration.</td>
</tr>
<tr>
<td>Small releases</td>
<td>The team delivers a working system in short intervals, at least at a monthly basis.</td>
</tr>
<tr>
<td>Metaphor</td>
<td>The system is defined by the use of a metaphor.</td>
</tr>
<tr>
<td>Simple design</td>
<td>The team uses the simplest designs possible.</td>
</tr>
<tr>
<td>Continuous testing</td>
<td>Software development is test driven, unit tests are performed when new code is developed.</td>
</tr>
<tr>
<td>Refactoring</td>
<td>The system is restructured continuously.</td>
</tr>
<tr>
<td>Pair programming</td>
<td>Two persons implement the code together, at one computer.</td>
</tr>
<tr>
<td>Collective code ownership</td>
<td>All team members can change any part of the code, at any time.</td>
</tr>
<tr>
<td>Continuous integration</td>
<td>Newly developed code is integrated in the code-base as soon as possible.</td>
</tr>
<tr>
<td>40-hour week</td>
<td>A working week consists of maximum 40 hours.</td>
</tr>
<tr>
<td>On-site customer</td>
<td>The customer is present and works with the team all the time.</td>
</tr>
<tr>
<td>Coding standard</td>
<td>The team has established rules for how to write code.</td>
</tr>
</tbody>
</table>

A more detailed description of all of XP’s practices is provided for instance by Beck [10]. There are practices that offer support to the development activity in more direct ways like the practice Pair programming,
where two engineers sit next to each other and develop software. A benefit of working in pairs is that the person who is not programming at the moment can review the other person’s code. This way more defects are probably found since it is easier finding defects in code that another person has written, compared to code that oneself has written as stated in XP. There are other practices like the practice Metaphor which helps in less concrete ways, by providing a metaphor for the system to be developed. The purpose is to improve communication between customer and programmers since the customer can avoid technical details but still talk about the system. XP advocates frequent use of unit tests. These tests are run every time new functionality is added, to make sure that the legacy code is still functioning correctly. Another aspect in XP is that it applies collective code ownership, i.e. every programmer has the right to make changes in the code at any time as he or she sees fit. Further, XP advocates an open workspace where pairs of programmers sit in open cubicles near each other to facilitate communication within the team. XP applies 40 hour-weeks to reduce the amount of stress on the engineers, since overtime is not productive in the long run. The practice Refactoring helps restructuring code while keeping the original functionality. Further, the practice Simple design is a mind set that helps the team to design as simple as possible, since it is common that future changes in requirements or functionality can mean that old and carefully planned designs are no longer applicable.
Chapter 3

RESEARCH MOTIVATION

3.1 Overview

The purpose of this chapter is to present and motivate the overall research problem that is addressed in the thesis. Further, the chapter includes an explanation of how and why the studies included in the thesis correlate to each other. The chapter ends with an overview of the studies’ separate subproblems and their associated research questions.

3.2 Research Problem

Software development processes provide structured and disciplined ways of developing software. Individuals, teams and organizations are supported in their efforts of producing high quality software by applying these processes. However, there is a difference in theory and practice when using a software development process since the process often needs to be adapted to the specific context where it will be applied. The transition from reviewing a process’ contents to applying it efficiently in practice may be cumbersome. However, this transition may be easier if there exists predefined support how to manage (e.g. introduce and adapt) the process in question. Figure 3.1 illustrates the purpose of such ”meta support” (i.e. support for improving the process, which in turn improves software development).

To further illustrate the relationship between support and process, a metaphor is provided. The process is a car which helps the driver (software developer) drive fast and steady to the destination (delivery of software product). Then the support, illustrated in Figure 3.1, is the tools that can help repair the car (process) if it breaks down. Thus, predefined support may improve the results of using a process. As explained in Section 3.3, the support may include any activity that can help improve process performance (i.e. increase quality of product and/or reduce costs or development time).

However, there is a lack of support which helps manage software devel-
CHAPTER 3. RESEARCH MOTIVATION

Figure 3.1: Relationship between support and process.

development processes. This is problematic since if such support was available to a larger extent, the results of applying software development processes would improve. The main reason for the lack of support may be that in order for the support to be valid, it should be based on empirical results from using software development processes. For example, Abrahamsson et al. [6] and Karlström [54] state that there is an urgent need of empirical studies for evaluating the effectiveness and possibilities of using different software development processes in various contexts. An increased understanding of how to adapt and use processes in different contexts is a sound basis when developing support for managing software development processes. In this thesis the problem of managing processes is divided into two subproblems:

- **The lack of support.** There is not much support available that is based on empirical data for most software development processes. Thus, in most cases there does not exist means for managing these processes efficiently.

- **The applicability of support.** There exist problems associated with applying support for managing processes. Further, there exists a limited amount of knowledge regarding which contexts the support can be applied in.

As mentioned in Section 1.2, the research in this thesis develops support for two different process types, agile and plan-driven methods. The first subproblem (i.e. the lack of support) is addressed in both the research that concerns agile and plan-driven methods. There exist little support that is based on empirical findings for agile methods. Further, although PSP provides means for collecting and analyzing process data for the software engineer, there are other software development contexts that may receive similar support which PSP provides. The second subproblem (i.e. the applicability of support) is addressed in the research that concerns plan-driven methods. The experience of the author of this thesis and earlier PSP research for instance conducted by Morisio [65] have indicated problems with
using PSP in practice, thus a solution for increasing the applicability of PSP is presented. The overall research problem addressed in the thesis is formulated as:

*There exists a limited amount of support that is based on empirical data for software development processes, which has a negative impact on the management of software development processes.*

This research problem is considered important, since an increased understanding how to support processes in different software development contexts may improve software development. As mentioned in Section 3.3, can support cover all kinds of activities that may improve software development. Further, there are numerous processes that can receive support. Thus, it is necessary to delimit the area that the research problem addresses. First, the agile method XP and the plan-driven method PSP were used when developing support for the process types. A more detailed motivation for this choice is provided in Section 3.3. Further, technological aspects are not addressed when developing support for managing software development processes. Surely, the introduction of a configuration management system for instance would probably help an organization manage its files and documents better. However, these kinds of issues are outside the scope of the research presented in the thesis. The thesis addresses support for managing software development processes based on human behavior.

Thus, the research in the thesis addresses the overall research problem defined above by developing support for agile and plan-driven methods. This overall research problem is addressed in different ways by the studies included in the thesis. Section 3.4 includes an overview of these studies’ problems and the associated research questions.

### 3.3 The Studies’ Relation to the Research Problem

This section motivates the studies included in the thesis, and how they correlate to each other. The term support is not unambiguous, thus it is motivated to explain its meaning in this thesis. The purpose of the support is to help management of processes in various ways so that product quality is improved and/or costs and development time are reduced. This goal can be achieved in different ways, for example through improved team work or a detailed review (and improvement) of the software development process and so forth. The term support includes all of these approaches, thus defining all kinds of activities that may change existing processes to improve quality and/or reduce costs and development time.

The plan-driven method PSP and the agile method XP were used in the research when developing SPI support for the two types of processes. It is
motivated to ask why these processes were chosen, since there exist other processes to choose from. Both processes are prominent and addressed in various research articles. Thus, there is substantial interest in these processes which motivates the choice to do research on them.

Below, a motivation for each study in the thesis is provided regarding how the studies address the overall research problem of this thesis.

**Study 1: Code Review Efficiency.** One of the objectives of this thesis is to develop support for plan-driven methods. The intensive process data approach inherent in PSP is used as a basis when developing the support. To develop support that is valid and resembles the approach in PSP, it was deemed necessary to gain experience how to collect and analyze process data when using a plan-driven method (i.e. PSP). Study 1 addresses this issue as process data gained from applying PSP was collected and analyzed in a smaller pilot study. This experience was used when developing support for plan-driven methods in Study 2 and Study 3. Study 1 is presented in Chapter 5.

**Study 2: SPI Support Based on Non-Technical Criteria.** The support (i.e. the mechanisms for collecting and analyzing process data) already exists in PSP. However, there are numerous roles in software development projects which are in need of SPI support similar to the support the PSP provides for the software engineer. Thus, the type of support that PSP provides should be applied in more contexts, thus providing SPI support in more areas in software development. Study 2 addresses this issue by developing support for a different software development context. Study 2 is presented in Chapter 6.

**Study 3: Individual SPI Support in RUP.** Another problem with the plan-driven support based on PSP, concerns the applicability of the support. There is an excessive amount of training and efforts required to efficiently manage PSP. This viewpoint is based on the personal experience of the author of this thesis and mentioned in articles for example by Morisio [65] and Börstler et al. [14]. Study 3 addresses this issue by investigating if and how the applicability of the support for managing PSP can be improved. Study 3 is presented in Chapter 7.

**Study 4.1: Quantitative Measures on Agile Development.** This study investigates quantitative results when introducing an agile method\(^1\) to a complex software development environment. A team used the process during eight months, and empirical data were gathered at regular intervals during this time period. Statistical analysis was

---

\(^1\)In Study 4.1 to Study 5, the term agile method means a process based on XP.
performed on the quantitative data. Thus, the team’s performance was measured from an objective perspective. This information is useful since organizations can, by comparing quantitative results to how a process is introduced, realize what parts of the process that provide most benefits (quantitatively speaking). Study 4.1 is presented in Chapter 9.

**Study 4.2: Qualitative Measures on Agile Development.** This study addresses the team’s experiences regarding how the agile method was introduced and adapted. Study 4.2 provides a structured evaluation based on qualitative data regarding the team’s perception of the agile method. This information is useful as potential problems regarding the introduction of the process may be identified. Study 4.2 is presented in Chapter 10.

**Study 4.3: Agile Collaboration Between Team and Organization.** This study investigates whether the introduction of the agile method improves collaboration between the team and its surrounding organization, seen from an organizational perspective. The purpose is to recognize the opinion from the surrounding organization, since an improved understanding in this issue may help realize how the agile method may be introduced to improve collaboration between the team and its surrounding organization. Study 4.3 is presented in Chapter 11.

**Study 5: A Survey on Agile Development.** This study addresses organizations’ experiences of introducing agile methods. A survey was conducted regarding how, why and in what types of organizations agile methods had been introduced into. The purpose is to gather information from a larger population regarding experiences and results of introducing agile methods. This improves accuracy and validity of the drawn conclusions regarding this issue. Study 5 is presented in Chapter 12.

The thesis addresses problems with support for agile and plan-driven methods. For plan-driven methods the problem concerns both the lack of support and the applicability of support. These problems are addressed partly by transferring the support structure based on PSP to a new software development context (Study 2), and partly by presenting a solution how to increase the applicability of the support provided by PSP (Study 3). For agile methods the problem concerns the lack of support. This problem is addressed by developing support for introducing agile methods based on XP into organizations (Study 4.1 to Study 5). The introduction of agile methods was chosen to receive the support since it may affect the future usage of a process. Further, an important aspect is how much of agile development
that the developed support should cover. Obviously, a full coverage as is the case with PSP is preferred (i.e. from design to testing). However, this option was not feasible to apply due to time limitations. Thus, the developed support concerns the activity of introducing processes based on XP into organizations. The support is developed in form of guidelines for how to introduce processes based on XP into organizations. The motivation to develop the support in form of guidelines is based on the fact that the introduction of a process is an important and complex activity. The way a process is introduced does not only concern the team that will use the process, but also the surrounding organization is affected as confirmed from the results of Study 4.1 to Study 5 but also mentioned for instance by Murro et al. [67]. Guidelines are regarded as an appropriate form to support this activity since they provide concrete advice how to address important issues. Further, they can also be applied in different ways depending on the specific needs of an organization. This property is essential since each organization is unique and develops software in unique ways.

3.4 The Studies’ Problems and Questions

The research in the thesis addresses the overall research problem defined in Section 3.2. This is done by conducting the studies included in this thesis as explained in Section 3.3. Each study has its own research problem and question. These problems are means to address the overall research problem.

The studies’ research problems and associated questions are presented below. The first three problems address support for plan-driven methods and the last two problems address support for agile methods. After each problem follows its associated question and a motivation of the importance of answering the question. Problem 4 was addressed by conducting a case study where a process based on XP was introduced to a team in a large software development company. The term based on XP is used since the purpose of the case study was to investigate how XP was introduced, adapted and used under real circumstances in the industry. This had the effect that parts of XP were either adapted or not introduced depending on the needs of the team. The case study (i.e. Study 4) included 3 studies that were conducted in parallel, thus the need to have several research questions.

**Problem 1 (Study 1).** There is a lack of knowledge how to conduct efficient code reviews at an individual level. The question for problem 1 is:

*How does the application of the criteria: 1) level of detail, 2) update and 3) LOC/hour affect code review efficiency?*

Humphrey [41] identifies three different criteria that affect the result
of code reviews. However, it is not clear what the criteria’s order of precedence is regarding positive impact on code reviews. In the study, PSP is used as an empirical framework to explore these criteria’s impact regarding this issue.

**Problem 2 (Study 2).** There is a lack of knowledge how and if non-technical criteria can provide SPI support in a software development context. The question for problem 2 is:

*To which extent can the application of non-technical criteria provide SPI support in a software development context?*

Results by Humphrey [41] indicate that software engineers receive thorough support in form of the PSP which supports improvement of so called hard factors that can be objectively quantified. Non-technical criteria on the other hand, are characterized as being difficult to quantify and thus control. Since non-technical criteria are difficult to quantify, it is reasonable to ask whether the application of non-technical criteria can provide SPI support.

**Problem 3 (Study 3).** There is a lack of knowledge how the software development process RUP [57] can receive SPI support at an individual level. The question for problem 3 is:

*How can SPI support at an individual level be introduced into RUP?*

RUP is a software development process which helps organizations to develop software in a structured and disciplined way. However, it lacks SPI support at an individual level. To address this issue, PSP was incorporated to RUP.

**Problem 4 (Study 4).** There is a lack of knowledge concerning the results of introducing agile methods in maintenance and evolutionary software development environments. Question number 1 for problem 4 is:

*Does the introduction of an agile process to an evolutionary and maintenance software development environment improve software development performance?*

It is important to increase the empirical body of knowledge concerning results of introducing agile processes in various contexts, as mentioned by Abrahamsson et al. [6] and Karlström [54]. To determine if the introduction of a process improves software development in an organization, it should be rigourously and empirically tested as stated by
Pfleeger and Hatton [74]. The question addresses this issue by providing quantitative empirical data on the effects of introducing an agile process based on XP into an organization.

Question number 2 for problem 4 is:

What are the experiences of introducing an agile process based on XP to an evolutionary and maintenance software development environment?

The question concerns how a team perceived the introduction and adaptation of an agile process. This is important to address since the answer to the question may identify problem areas when introducing an agile process based on XP to an evolutionary and maintenance software development environment.

Question number 3 for problem 4 is:

How does the use of an agile process affect the collaboration in a software development organization?

As stated by Murro et al. [67], the introduction of a process in an organization affects more people than just the software engineers. To successfully introduce an agile method it is important to recognize how the surrounding organization perceives the introduction of the process. The answer to the question may help understand how agile development affects collaboration in organizations, seen from an organizational perspective.

Problem 5 (Study 5). There is a lack of knowledge concerning the introduction of agile methods in various software development contexts. The question for problem 5 is:

What are the results of introducing agile methods based on XP into software development organizations?

The question addresses how agile methods based on XP are introduced to a number of organizations. There is a limited amount of research conducted on this subject, which forces organizations to make uninformed decisions how they should introduce and adapt this kind of process.

The problems are addressed in the thesis as follows. Problem 1 is addressed in Chapter 5. Problem 2 is addressed in Chapter 6. Problem 3 is addressed in Chapter 7. Problem 4 is addressed in Chapters 9, 10 and 11 and problem 5 is addressed in Chapter 12.
Chapter 4

RESEARCH METHODOLOGY

4.1 Overview

The purpose of this chapter is to summarize and motivate the kind of research strategies, methods and data collection techniques that were applied in the studies presented in this thesis. The chapter starts with an overview of different research strategies and methods in general. This information is useful as it provides a research background, when the research presented in the thesis is discussed and motivated later in greater detail.

4.1.1 Research Strategies

A research methodology or strategy could be described as a collection of assumptions and considerations that are applied when performing research, or more informally as the broad orientation taken in addressing various research questions as stated by Robson [84]. Use of an established research methodology provides a procedural framework where the researcher uses proven strategies for conducting research in a scientific manner, which increases the reliability of the research results. Further, as mentioned by Robson [84] it is important to select a research strategy that is appropriate for answering the specific research questions. Depending on the amount of research performed on a subject, the research strategy can either be exploratory, explanatory or descriptive. An exploratory research strategy means basically to find out what is happening in contexts or situations where there has been little research conducted. Explanatory strategies differ from exploratory by seeking to explain a problem or situation, by identifying relationships between different aspects of the phenomenon studied. Explanatory strategies are often applied in areas where there already exists a sufficient amount of research, which makes it possible to study a phenomenon in greater detail.
The purpose of descriptive research strategies is to as accurately as possible describe a phenomenon of some sort. In contrast to exploratory and explanatory strategies, a descriptive strategy is limited to only portraying a situation and may not identify what causes a specific behavior.

Research can be classified to be either deductive or inductive to its character. Deductive research strives to draw conclusions based on recognized facts, or in mathematical terms if $p \rightarrow q$, and $q \rightarrow r$, then $p \rightarrow r$. Thus, deductive research uses already established theories as a starting point when formulating research problems. An inductive research view on the other hand draws conclusions based on local observations, going from the specific to the general where collected data is used to build theories. As the research was mainly explorative and descriptive in its character, where conclusions emerged as data was analyzed, it is justified to classify the research view being more inductive than deductive.

There exist different epistemological assumptions on what knowledge is and how it can be obtained. Based on these viewpoints, research tend to be classified as either quantitative or qualitative in nature. The positivist epistemology, who favors the quantitative paradigm, is based on the assumption that knowledge is objectively gained by collecting and analyzing numerical data. This research paradigm has its origin in the natural sciences such as physics. The purpose is to establish universal laws based on quantitative empirical findings, as stated by Robson [84]. The positivist epistemology has an objective explanation of science as a goal. It does not support the assumption that knowledge can be obtained based on non-numerical data. For instance, a scientific theory can not be based on human observations as each observation is subjectively biased of the unique experience of an individual. The subjective experience fogs the view of the objective truth, according to the positivistic view.

The qualitative paradigm, on the other hand, argues that knowledge may indeed be based on the subjective experience of humans. The purpose of qualitative research is to help understand people’s behavior within their social and cultural contexts, as mentioned by Robson [84]. Constructivism, or interpretive research, which it is also referred as by Robson [84] is a qualitative research paradigm which does not approve of the notion that there exists an objective reality. It argues that reality is a social construct, where reality is understood by investigating people’s attitude and behavior towards a phenomenon under study. Ethnography, for instance, is a qualitative research strategy which is characterized as interpreting and explaining how a group or community live and experience their lives and their world, as stated by Robson [84]. Typical features of an ethnography study are that the researcher lives with the group or community and it takes long time, in some cases years, before any results can be obtained. These aspects made the ethnography approach unappropriate to apply in the research.

The main differences between a qualitative and a quantitative research
strategy are that qualitative research enables a detailed view on social inter-
play and is usually more flexible in the sense that during a study it is possible 
and encouraged to change research stance if there is a need to. Quantitative 
research, as argued by Robson [84], has a more formal and strict approach 
to research design where a hypothesis is stated and then tested by use of 
statistical analysis. Further, quantitative research is usually based on larger 
samples thus increasing the probability that the sample is representative of 
the population. The main part of the research presented in the thesis is 
based on a qualitative paradigm. However, it should be noted that a mixture 
of quantitative and qualitative research approaches and data collection 
techniques were used, as described in Section 4.2.

4.1.2 Research Methods

A research method identifies how to actually perform the intended research. 
When choosing a research method, concerns should be made to the research 
environment and purpose of the research. Thus, the choice of research 
method should be based on the type of research strategy that is selected. 
There exist a number of different research methods to apply in a software 
engineering research context. Below, a description is provided on the meth-
ods that were applied in the research, except the literature study which is 
so basic in its content that a description of that method is not included.

Experiment. The experiment is a research method used in quantitative 
research. Experiments can be applied when using an explanatory ap-
proach to the research, when the aim of the research is to explain 
relationships between different variables of a phenomenon. Experi-
ments are characterized by Robson [84] as assigning participants to 
different conditions, manipulating one or more independent variables, 
measuring the effects of this manipulation and during the experiment 
controlling all other identified variables. Experiments require a strictly 
controlled research environment, which may be difficult to create when 
investigating software development in industrial settings.

Survey. Another research method used in both quantitative and qualitative 
research, is the survey for instance described by Biemer and Lyberg 
[11]. A survey is characterized by the use of a fixed, quantitative de-
sign. Further, data is collected in a small amount and in a standardized 
form, from a relatively large number of individuals in an attempt to 
validate local findings on a broader public. Thus, it is important to 
select representative samples of individuals from known populations.

Case Study. A research method in the qualitative research arena that does 
not require this type of controlled research environment as advocated 
by the experiment research method, is the case study. The case study
CHAPTER 4. RESEARCH METHODOLOGY

method concerns development of detailed, intensive knowledge about a single case or of a small number of related cases, as described by Robson [84]. Typical features of a case study are selection of a single case of a situation, individual or group or other aspect of interest and study the case in its context. Further, collection of information can be done in a number of different ways such as interviews and observations but also through measurements of a quantitative character, thus allowing both subjective and objective data collection techniques.

Grounded Theory. Grounded theory, described by Glaser and Strauss [36], is a research method where the purpose is to generate theory from data collected during a study. Grounded theory is useful in new areas where there is a lack of knowledge to describe occurring events. Theories may change and evolve as data is gathered and analyzed. Further, grounded theory is applicable in social science research. Grounded theory is mainly used in qualitative studies involving human behavior and interaction which software development consists of. Although considered a qualitative research method grounded theory allows both qualitative and quantitative data collection techniques. Further, grounded theory has no ambition to achieve statistical generalizability by using random samples. The purpose is to understand a phenomena by choosing samples that provide insight to the phenomena studied.

4.2 Description of the Research

4.2.1 Overview

The purpose of this section is to motivate and explain the research methods and data collection techniques used in the studies presented in the thesis. First, an overview is provided of the strategies, methods and data collection techniques used in the thesis. Then, a detailed account of the research choices made in each study is provided. The studies are traversed in the same order as presented in Section 3.3. Table 4.1 presents the chosen research strategies, methods and data collection techniques for the studies presented in the thesis. As is seen in the table, a number of different research approaches were used in the thesis. This depends on the fact that the studies differed in purpose which required use of different research strategies, methods and data collection techniques.

Study 2 and Study 4 address elements of grounded theory in the sense that research was performed in areas where there has not been much research performed. The collected data was reviewed several times and new conclusions could emerge which contradicted earlier conclusions. The concept of grounded theory was embodied in the research when developing SPI support for the BPA role, presented in Study 2 as the author of this thesis
Table 4.1: Strategies, methods and data collection techniques used in the research.

<table>
<thead>
<tr>
<th>Study</th>
<th>Strategy</th>
<th>Method</th>
<th>Data Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Descriptive, explanatory</td>
<td>Experiment</td>
<td>Quantitative</td>
</tr>
<tr>
<td>2</td>
<td>Descriptive, explorative</td>
<td>Literature survey</td>
<td>Quantitative, qualitative</td>
</tr>
<tr>
<td>3</td>
<td>Explorative</td>
<td>Literature survey</td>
<td>Qualitative</td>
</tr>
<tr>
<td>4</td>
<td>Descriptive, explorative</td>
<td>Case study</td>
<td>Quantitative, qualitative</td>
</tr>
<tr>
<td>5</td>
<td>Descriptive, explorative</td>
<td>Survey</td>
<td>Quantitative, qualitative</td>
</tr>
</tbody>
</table>

did not have any prior knowledge how the BPA role could be supported and the support emerged as BPAs were interviewed and literature was read in the area. The research was similar to grounded theory as the support for the BPA role gradually emerged as more and more data was analyzed. An important difference in how grounded theory was used in Study 2 from how grounded theory is supposed to be applied is that a literature survey was conducted, which contradicts the fundamentals of grounded theory since the researcher is supposed to have an open and unbiased research opinion which may be affected if performing a literature study. The research in Study 4 had some aspects in common with grounded theory in the sense that there did not exist any knowledge regarding the introduction of the agile method in this kind of environment, prior to the introduction of the process. Conclusions emerged as data was collected and analyzed. However, the research in Study 4 was based on underlying theories on what the expected outcome of the studies would be, which contradicts the nature of grounded theory.

4.2.2 Study 1: Code Review Efficiency

Study 1 applied an experimental approach, where different criteria’s impact on code review efficiency were investigated. An experimental approach was motivated since it was necessary to control different parameters in the research context. The experiment was conducted in an academic environment where PSP was taught to a number of students. The students who participated in the study did their last year on a master of science degree in software engineering. The motivation of using students as study participants were mainly based on two aspects. First, it is our experience, and others as well, that it is difficult to transfer PSP into industry since the effort to learn PSP is substantial which requires both time and effort, for instance mentioned by Humphrey in [44]. Thus, it is not a trivial task to
get people working in the industry to learn all of PSP. However, this study needed process data from people using PSP and this could be done in an academic context. Second, in many cases it may be preferred to use people from the industry in software engineering research as they represent the real world so to speak. However, in this case different criteria’s application were investigated concerning how code reviews performed at an individual level could be improved. This is a single and isolated activity, in the sense that external factors such as team work and so forth do not affect the result of the review. When a person conducts a code review, he does it alone in an undisturbed context, and this environment can be fulfilled in the academia as well as in the industry. Thus, it is the opinion of the author of this thesis that the study could be conducted with the use of students and still achieve valid results.

Data was collected from seven students. There were more students that signed up for the course, but due to different reasons some of them did not complete the course, which reduced the number of study participants. The students needed to complete 10 exercises using PSP in order to complete the course. However, the code review activity was not included in the process until exercise 7. Hence, the results are based on analyzing four assignments for each student. As they progressed through the course and did their assignments, the students stored process data of their performance electronically. Thus, their data could be analyzed after the assignments were completed. This approach had the advantage that the students were not interfered when they did the code reviews. This was considered important since if the students had been monitored in any way they may have changed their behavior, consciously or unconsciously, and this was not a desired effect since the ambition was to minimize the number of validity threats to the study. Further, the students were assured anonymity regarding their performance in the study. This way, they did not have to think about whether someone else would judge and criticize their performance.

4.2.3 Study 2: SPI Support Based on Non-Technical Criteria

The target population was defined to be Business Process Analysts, BPAs, from the industry since it was important to evaluate the meaningfulness of the developed SPI support based on input from people who work with these kind of factors in real environments. This increases the validity of the results. When selecting study participants no sophisticated sampling technique was used, due to time and economical constraints. Instead, available BPAs were interviewed in the local region. This approach could be summarized as convenience sampling described by Wohlin et al. in [97]. However, to ensure that the chosen BPAs were representative of the target population, they were subjectively assessed through interviews before participating in the study. The study participants were seven in total. The ambition was to include
people with different amounts of experience of analyzing business processes since a multi-faceted view may indicate whether the developed SPI support is better used in certain contexts and so forth.

The interviews were of the type open-guided interview, as described by Lantz [58]. Open-guided interviews are useful when the interviewer has limited knowledge about the research area in question. These kinds of interviews have a wide research question, where the respondents can broadly answer and reason about the question. This type of interview was considered appropriate in the study due to the above mentioned aspect.

The contents of the SPI framework, that is the BPA’s means to collect and analyze his/her process data, were based on the PSP framework defined by Humphrey [41]. Earlier research has shown that software engineers who use PSP, may improve their software development processes. These results motivated a similar structure in the BPA framework. Obviously, it is important to develop framework contents that are useful and valid for the BPA role. This was achieved by using the Goal Question Metric, GQM, technique for instance described by Berghout and Solingen [91], where goals identified from interviews with BPAs were used. Further, the framework elements were graphically displayed in order to better understand the couplings between various templates and metrics and so forth. This way, a better structure of the framework, that is when and where the framework elements would be used, could be achieved.

A questionnaire was used in the assessments where each answer to a question was graded with five alternatives. The ordinal and absolute scale types were used on the questionnaire, because in the ordinal scale type an ordering among classes exists which was useful as the BPAs were asked to grade subjectively the amount of support certain framework elements provided, ranging from strongly hinder the BPA in his/her work, to strongly support the BPA in his/her work. Further, the absolute scale was chosen since it was important to count how many times each alternative was chosen.

The BPAs were interviewed twice. The second round of interviews differed from the first round of interviews as the questions addressed usage of the framework, whereas the first round of questions addressed the BPA’s role in software development projects. Thus, the purpose of the second round of interviews was to gain additional information about the framework’s support for the BPA’s work in a software development context.

4.2.4 Study 3: Individual SPI Support in RUP

A motivated question to ask is why RUP was chosen to receive SPI support at an individual level, since there are other processes or standards used by organizations when developing software, for instance SPICE [51] or the CMM [73]. RUP has defined roles, which SPICE does not have, similar to the role that PSP supports and these roles lack the kind of support that
PSP offers. The other option, for instance to incorporate PSP into CMM (which has roles) would be more difficult since CMM does not provide clearly defined roles which is vital when incorporating the contents from PSP. Thus, the condition when introducing SPI support based on PSP into a process at an organizational level, is that the process in mind should have clearly defined roles.

After reviewing RUP, 5 roles were chosen to receive SPI support from PSP. The roles were chosen after comparing their profiles against PSP’s profile. Each role covered parts of PSP, for instance one role that was chosen was a tester role and testing is one part of the role that PSP supports. It should be mentioned that some contents of PSP, such as the process improvement proposal template which is simply a mean to record ideas or comments and so forth when developing software, are general in nature and can be successfully applied in a number of environments. PSP was incorporated to five selected roles in RUP which could apply the more context-dependent content of PSP. However, other roles in RUP may find parts of PSP useful.

Before incorporating PSP into RUP it was necessary to compare the processes’ different software development contexts, to know where in RUP the PSP material would be disseminated. To perform this comparison software development terms used in RUP were compared with terms used in PSP. However, when comparing RUP with PSP it was noticed that RUP and PSP sometimes used different labels on terms that were semantically equivalent, or sometimes did terms in RUP and PSP have the same label but were semantically different. This motivated a more detailed comparison of the terms used in RUP and PSP, after it was decided what parts of PSP that would be allocated to each of the five roles. The procedure of allocating PSP contents to the roles in RUP was based on our knowledge of PSP and RUP. Thus, the decisions regarding this issue were subjectively taken. Another option would have been to interview people working with PSP and RUP to receive their opinions on how PSP could be incorporated into RUP. However, few people use RUP and especially PSP according to our knowledge which makes it difficult to get people to interview. Thus, this option was not chosen.

The intention of the study was to provide a way how to incorporate PSP into RUP. Each organization and environment has its own characteristics where the PSP material need to be adapted accordingly. Thus, before using PSP in RUP it should be adapted to the specific context where it will be used. To address this issue, guidelines were developed which organizations can use as support when introducing and adapting PSP in RUP.

4.2.5 Study 4: A Case Study on Agile Development

The case study method was applied when investigating the effects of introducing an agile method into an organization. The agile method was used
by a team which consisted of six developers and one team leader. The ambition was to investigate various aspects of introducing the agile method in a real software development environment, seen from several perspectives. This meant that the team that was introduced to the process and its surrounding organization were observed and measured while using the process in their work. Thus, there was no intention of controlling the research environment as the purpose was to introduce the process under normal working conditions for the team. Further, as aspects of using the process were investigated from several perspectives, a number of different data collection techniques were used. These factors, that is a detailed study of a team in its natural context and the use of several data collection techniques, motivated the choice of a case study as the research method in this study.

The case study consisted of 3 separate studies, that were conducted in parallel. One of the studies investigated software development results when the team applied the agile method. Thus, quantitative data was collected and analyzed each week. The data was collected and analyzed weekly since it was considered a manageable pace to sustain as 3 separate studies were performed at the same time. Further, the team was informed of their progress each week, based on an analysis of new data, as a motivator to improve their performance. Both statistical tests and descriptive statistics were applied on the data. Statistical tests were applied as one purpose was to assess whether the team improved their results after compared to before the agile method was introduced. Here, the statistical tests were used as they could state whether any improvement was statistically significant, that is an improvement is probably not a result of chance. Descriptive statistics were applied since trends in the team’s achievement wanted to be identified, and these trends were identified when a majority of the collected data were of the same value, which is calculated by use of descriptive statistics. Further, these trends were shown to the team each week as an indication of its performance, since the results were easy to understand compared to the more advanced approaches used in statistical tests. The metrics were defined after recognizing the organization’s goal with the study. It is vital that the metrics are valid and based on a theoretical foundation, which was fulfilled from the researcher side. However, it is also important that the industry side feels involved when defining a study as this will raise its commitment and involvement in the study. Thus, the metrics which addressed productivity and estimation issues, were based on a joint understanding between researcher and industry.

The team that was introduced to the agile method was chosen due to three factors. First, it worked solely against a single software system which allowed us to investigate the impacts of using the method as no other teams worked on this system. This situation was unique in the organization as no other team worked solely against a system. Second, the team had stored data on its performance during a long time which allowed comparisons of
process data before and after the method had been introduced. This provided an opportunity to investigate whether the agile method helped the team improve its software development performance. Third, the team was interested in learning and using the agile method, which meant that no one had to be forced to participate in the study. As one purpose of the study was to investigate how the process was introduced and adapted in a real environment without any requirements from a research perspective, it was allowed for the team to use as much or little of the process as it liked. Further, the contents of the method could be adapted as the team thought appropriate.

This study addresses the team’s experiences regarding how the agile method was introduced and adapted, and what effects these adaptations had on the team’s performance. The data collection techniques that were used were interviews and observations. These types of techniques are commonly applied in qualitative research as mentioned by Robson [84]. Interviews were used since they provide information based on verbal communication and it was essential to talk to the team in order to understand their motivations why certain practices were adapted in different ways. Observations were used since they offer additional information that may not emerge in interviews, such as how the team physically organizes its software development environment or the expression of different emotions that may indicate joy or frustration and so forth. Thus, it is our opinion that the combination of interviews and observations improved the validity of the results. The interviews were semi-structured, where the respondents replied on rather unstructured questions. This approach, where the respondent was encouraged to reply freely was useful since it was important to receive as much information as possible regarding how and why the team adapted XP as it did. The observations on the team were of the none obtrusive kind which means that the researcher does not interact with the observed party. This approach was chosen since the purpose of the study was to investigate the team when it applied the agile method in its natural environment. Both the interviews and the observations underwent a qualitative analysis, described by Lantz [58]. This analysis approach is similar to the analysis steps encountered in grounded theory, in the sense that there is an iterative procedure of critically reviewing the collected data in order to find relationships or patterns in the data that may help identify valid conclusions. The team was asked to subjectively rank the introduced practices with regard to how easy it was to introduce the practices and how much positive effect the introduction had on the team’s performance. This information was considered useful as it is our experience that many organizations only introduce a subpart of XP’s practices. The introduced process was compared to XP, with respect to how the processes were used. The identified differences in applying these processes were recorded. Further, differences in how the team developed software prior to the introduction of the agile method, compared to the way the team developed software after the method had been introduced were
identified. The reason to conduct these detailed comparisons was to understand how the team had changed its software development process, and what the results from this change was.

The last of the three studies that were conducted in the overall case study addressed how the surrounding organization perceived whether its collaboration with the team changed as a result of the team started using the agile method. The data was collected in form of interviews conducted with the surrounding organization, that is the conclusions from the study were based on an organizational perspective since the purpose was to investigate the organization’s opinion regarding this issue. The interviews were semi-structured, with the same motivation as mentioned above. Observations were not used since the people in the organization that collaborated with the team were busy and did not work with the team closely, which made it difficult to observe them when they collaborated with the team. Further, as their contact with the team was scarce the amount of data collected from observations would also have been scarce which affects the validity of the results in a negative way. The data was analyzed by using the same qualitative data analysis technique described by Lantz [58], as mentioned above in the presentation of the former study.

4.2.6 Study 5: A Survey on Agile Development

The survey method was used in Study 5 which investigated organizations' experiences of introducing agile methods. The purpose of the study was to gain an improved understanding of how organizations introduce and adapt these kind of methods, and in which environments they are used. Based on the experience from Study 4 and reading relevant literature such as [56], [38], [39] and [86] questions were formulated to address issues of interest regarding the introduction of processes based on XP.

The target population was software development organizations from Sweden and Norway. The motivation for this was that these countries have similar cultures, beliefs and values in the society and this is also true for the software development community. These similarities in the cultures minimize any cultural aspects that may bias the results, which advocated sampling from these countries. Further, the survey addresses the case study presented in Study 4, where an agile method was introduced to a Swedish organization and since the questions in the survey were partly based on results from the case study, it was natural to direct the survey at Scandinavian organizations. There were no requirements on the characteristics of the organizations as the intention was to include as many organizations as possible and include a large variety of organizations. This way, it would be possible to identify if some introduction or adaptation issues only concern certain environments or types of organizations. A detailed background of the survey participants was acquired since an improved insight of their software
development history helps interpret the results and better understand why these results emerged. This background information increases the reliability of the results.

The study used an electronic mail survey. By using mail as way of communication, it was easy for us and the organizations to communicate (e.g. difficulties in interpreting a question). Interviews were not conducted since it would have been problematic to conduct interviews as the organizations were dispersed over a large geographical area. Also, since the goal was to include as many organizations as possible it may have been difficult to find available time to interview all of the organizations that participated in the survey.

Before embarking on the main survey, a pilot survey was conducted which the team from the case study in Study 4 participated in. The motivation for conducting a pilot study is to identify and remove any aspects of the survey which may reduce the quality of the survey and its associated results as mentioned by Biemer and Lyberg [11], such as unnecessary complicated sentences and so forth. Further, a research colleague reviewed the design of the survey from an agile process point of view. This helped realize if some important aspects related to introduction and adaptation of agile methods had been neglected in the survey. This activity is referred to as an expert review and is described in more detail by Biemer and Lyberg [11].
Part II

Support for Plan-Driven Methods
Chapter 5

STUDY 1: CODE REVIEW EFFICIENCY

5.1 Background and Problem

To locate and fix a defect in test phase versus prior to test is a ratio of about 6:1 as reported by Humphrey [41], defect removal costs are highest during test and use. Therefore in order to save time and effort, defects should be identified and removed as early as possible in the software development process. Conducting reviews are a way of finding and fixing defects in software programs. Reviews are the principal method of validating the quality of a process or product as mentioned by Sommerville [92]. The yield is the percentage of defects found in a review at a specific phase. However, in addition to a high yield time spent in a review is an important factor. Time spent in a review together with the yield metric provide information not only on the percentage of removed defects but also on the amount of time spent in a review. Thus, when reviewing a software product, a high yield should be associated with minimum time spent in review activities.

In PSP, a personal review is done by the software engineer trying to locate and fix defects prior to test phase. The software engineer examines his/her work, e.g. code or design, using a checklist based on personal historical defect data. The review activity is not an automated process. The success of a review depends on how well prepared and motivated the reviewer is. Humphrey [41] has identified three criteria that affect code review efficiency, but it has not been investigated these criteria’s order of precedence regarding positive impact on code reviews. This study addresses this problem.

PSP was used as the context for the study, since PSP is a defined software process which means that similar studies can be conducted and results compared since the context for the studies is the same, as pointed out by Wohlin [96]. Further, PSP has a number of metrics which can be used for
empirical studies. These aspects make PSP well suited as a study context for empirical research in software development.

Section 5.2 includes related work. Section 5.3 describes the design of the study. In Section 5.4, the study is presented. The economics of defect removal are discussed in Section 5.5, and Section 5.6 includes concluding remarks.

5.2 Related Work

This work concerns using PSP in a course context as a framework for empirical research in software development. The PSP course structure applied by the SEI involves ten programming exercises, where students apply PSP. One problem with this is that different students have different amounts of knowledge in a programming language. Thus, metrics like productivity, injected defects and so on also depend on programming skills, and not only on PSP skills. This issue is discussed by Höst et al. [40]. The authors note that improvement effects related to programming skills are most dominant in the first exercises. However, Wohlin [96] also discusses learning effects based on knowledge of programming skills, where it is argued that initial degree of programming skills does not significantly affect the combined result of all ten PSP assignments. Taking these results in consideration, a safety precaution in a study may be to skip PSP assignment results from the first exercises, if students participating in a PSP course have considerably different amounts of programming skills.

A study in learning PSP in an industrial setting was conducted by O’Beirne and Sanders [71]. The study shows an overall improvement in the software development process. The group average of yield (for a more detailed description of the yield term see Section 5.4) improved almost four times from exercise number one to ten, and the group average of Test Defects/KLOC was halved. The improvement in yield was achieved at no loss of productivity, it was about the same through all the assignments. The identification and removal of defects before compile and test phase, resulted in saved compile and test time. The compile and test time, as % of total development time, was reduced from 25% to 11%. The author of the paper remarks that it is desirable to find an optimum review rate to recover the best yield of defects.

Ferguson et al. [30] investigate the economics of defect removal, with or without reviews. They study shows that the number of defects found in test without reviews, is higher than with reviews. Humphrey [41] has published research material regarding optimum review rate where it appears that rates of 100 to 300 LOC per hour give the highest yields. Further, a metric for comparing defect removal efficiency in phases is defined. This way, the software engineer can realize in which phases his/her defect removal activities
need to improve most. More work have been done within the context of PSP, where the focus has been on reducing time spent in defect removal activities. The results presented by Stavros and Avratoglou [93], for instance, show a reduction in test time by 15% due to efficient reviews. Review checklists were used based on defect profiles of the software engineers.

5.3 Design of the Study

The study was performed on a class of students taking a PSP course. The course’s structure was the same as for the course given by the SEI [41]. The PSP course consisted of ten lectures covering PSP material and two seminars where the students’ process data were shown and PSP concepts discussed. In addition to this the course included ten programming exercises, where each exercise applies a PSP level. The first exercise applies a basic PSP process, whereas it is continuously enhanced to a more advanced process in the last exercise. Each exercise takes two to six hours to complete, and the last exercise requires more work than the others. The number of students attending the course were seven and the programming language was Java, which all students had good knowledge of. Data from all exercises were included in the study since we, apart from studying different criteria’s impact on code review efficiency, examined the defect removal situation with or without applying reviews. Defect removal activities without reviews, which the six first exercises address, is not closely related to any specific form or script in PSP. Therefore defect removal data will not be affected by a limited insight into PSP which is the situation at the beginning of the course. Design and code reviews require use of forms and scripts. However, these activities are not included before Exercise 7, and at that time the students were familiar with basic PSP elements, therefore no data was neglected.

5.4 Yield Management

Review yield refers to the percentage of defects in design or code at the time of the review that were found by that review, as defined by Humphrey [41]. In PSP, personal reviews are done where the software engineer examines his/her work using a checklist which is based on personal defect data. A high yield means less time spent in defect removal activities in later phases. However, a low yield may mean a waste of time since only a low percentage of any injected defects were identified and removed. Therefore, it is important to have optimal conditions for producing high yields. Humphrey, founder of PSP [41], identifies three criteria that affect the efficiency of a code review:

1. The level of detail of the checklist. If a checklist is not detailed enough, it will be hard to find defects.
2. *The update of the checklist with a new defect profile.* Over time, the software engineer may start inject new types of defects and/or stop inject old types. If the checklist is based on old defect material the reviewer will likely have a low yield percentage.

3. *The number of reviewed LOC/Hour.* LOC/Hour is an instant measure which enables the engineer to better manage his/her review process. Earlier PSP research, indicates that rates of 100 to 300 LOC per hour are associated with high yields.

As mentioned in Section 5.1, it is not clear how much each criterion affects code review efficiency. In the study, this issue is examined. Hence, the research question is formulated as:

*How does the application of the criteria: 1) level of detail, 2) update and 3) LOC/hour affect code review efficiency?*

Thus, the result of the study is an order of precedence regarding the criteria’s positive impact on code reviews.

### 5.4.1 Analysis

A boolean value is assigned to each criterion for the students. Then, the boolean values are compared against the students’ average yields, and it is deduced which criterion has the strongest influence. More advanced statistical tests were not considered since the number of data points is low which makes it difficult to state whether there is any statistical significance between the different ways of performing code reviews.

For instance, if the checklist has not been updated with a student’s new defect profile when the student performs the assignments the value False is assigned to that criterion for that particular student. Thus, by analyzing each student’s boolean values for the three criteria and considering their yield average, which is a measure of how well they performed in the code reviews, it can be concluded with logical reasoning which criterion has most positive impact on the code review activity. That is, logical in the sense that the criteria’s boolean values are compared with the code review results, and based on this it is concluded which criterion has most positive impact on code review efficiency. The decision whether to allocate the values True or False to a criterion is based on either a subjective or objective measure. For the criterion *the level of detail of the checklist* it is a subjective measure where the students’ checklists are reviewed and if the checklists are considered vague and allow different interpretations the boolean value False is allocated to the checklists in mind. On the other hand, if the formulations are precise the value is True. For the criteria *the update of the checklist* and *the number of lines of code reviewed per hour* objective measures are used since it can be
concluded whether the checklists have been updated from one assignment to another when students change their defect profiles. Further, by analyzing the duration of a code review and considering how many lines of code the code review addressed the number of lines of code reviewed per hour can be calculated. The criterion LOC/Hour is allocated a the value False if the review rate exceeds or falls short of the review rate interval recommended by Humphrey, namely 100 to 300 LOC/Hour [41]; else it is allocated the value True. The metric LOC/Hour could be used in the study since all students had very similar LOC counting standards. This way of analyzing is chosen since it allows a good overview of the students’ results and their different ways of conducting code reviews.

Table 5.1 shows the allocations of boolean values to the three criteria for each student. Further, the yield average (expressed in %) for each student is presented.

Table 5.1: Code review results.

<table>
<thead>
<tr>
<th>Student</th>
<th>Level of detail</th>
<th>Update</th>
<th>LOC/Hour (average)</th>
<th>Yield (average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>37.5</td>
</tr>
<tr>
<td>2</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>29.5</td>
</tr>
<tr>
<td>3</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>16.25</td>
</tr>
<tr>
<td>4</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>17.5</td>
</tr>
<tr>
<td>5</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>37.5</td>
</tr>
<tr>
<td>7</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>50</td>
</tr>
</tbody>
</table>

Below, the criteria’s impact on code review efficiency are discussed based on the students’ results.

Update versus LOC/Hour. All students had False on update. Earlier PSP research provided by Humphrey [41] shows that students taking a PSP course with these exercises had an average yield of about 70% where all students updated their checklists regularly, whereas this class managed an average yield of 36%. This indicates that update of checklist is important.

Student 2 and Student 3 had True on LOC/Hour, whereas Student 1 had False, but Student 1 had 8% respectively 21.25% higher yield than Student 2 and Student 3. However, Student 2 and Student 3 had injected more than twice as many defects than Student 1 which may make it more difficult to reach high yields as there are more defects to detect. Although recognizing this aspect, the difference in yield is far less than the difference regarding whether the checklists had
been updated or not. This indicates that the criterion Update is more important than the criterion LOC/Hour.

**Level of detail versus LOC/Hour.** A comparison between Student 2 and Student 3 (False, False, True) with Student 5 and Student 7 (True, False, True), shows that there is a total difference of 67.25% in yield, in favor of Student 5 and Student 7. It shows a significant difference in yield efficiency if the LOC/Hour is True and the Level of detail is True, compared to if the LOC/Hour is True and the Level of detail is False. On the other hand, if Student 4 and Student 6 (True, False, False) are compared to Student 5 and Student 7 (True, False, True) there is a total difference of 58% in yield, in favor of Student 5 and Student 7. When the Level of detail is True and the LOC/Hour is True, the yield value is significantly greater compared to if the Level of detail is True, and the LOC/Hour is False. These two arguments indicate that the criterion Level of detail, has somewhat stronger affect on the yield than the criterion LOC/Hour.

**Level of detail versus Update.** Student 1 (False, False, False) had 37.5% in yield, whereas Student 4 and Student 6 (True, False, False) had 17.5% and 37.5% in yields. Thus, although Student 4 and 6 used detailed checklists, they did not have higher yields than Student 1. Thus, the result indicates that the criterion level of detail has a lower impact on code review efficiency than the criterion update of checklist.

### 5.4.2 Summary

Figure 5.1 shows the students' review rates compared to yield values. The results support the recommended review rate of Humphrey, i.e. 100-300 LOC/Hour [41]. Although the study indicates that LOC/Hour seems to be the criterion that has least impact on review efficiency, it should not be neglected.

An update of the checklist seems to be essential if striving for high yields, but to save time the level of detail of the checklist also seems important. The research material shows a strong correlation of time spent finding and removing a defect, with the level of detail of the checkpoint. A vague formulation of a checkpoint provides little support when trying to find and remove defects. Using vaguely formulated checkpoints when reviewing is similar to finding a defect in test since the software engineer cannot identify the defect directly but has to find it, searching through a wider problem area.

When yield impact from the criteria that were identified by Humphrey are compared, the conclusions drawn apply under certain circumstances. If it is concluded that update for example has more impact than LOC/hour, that conclusion is based on our research data. What the conclusion would be with other research data is not known. Therefore, emphasize should be
5.5 Economics of Defect Removal

Identification of defects in design or code reviews takes less time than in test phase. Therefore, defects should be identified before a software product enters test phase, to save time. About half of injected defects are found by the compiler, and the rest must be found either by reviews or testing as reported by Humphrey [41]. Earlier PSP research conducted by Humphrey [41] shows that, for a modest-sized product, it requires five to ten programmer hours to find a defect in test, whereas reviews on average take 0.5 hours to find a defect. Further, PSP research indicates that software engineers normally inject 100 or more defects per KLOC in their code. Taking this in consideration, the following example shows the economics of defect removal when conducting reviews.

Effort without reviews. A product of 50 KLOC would enter test phase with around 50 or more defects per KLOC. This results in 2500 defects that must be removed in tests. If the removal of one defect takes about five to ten hours, and for the sake of reasoning the value 8 is used, it would require 20 000 or more programmer hours to get a defect-free product.

Effort with reviews. Earlier PSP research implies that when conducting reviews, on average 70% of the defects are found. A 70% review yield of 2500 defects are 1750. If the removal of one defect in review requires
0.5 hours, it means 875 hours to remove 1750 defects. The remaining 750 defects would have to be found in test phase at a cost of about eight hours each. The total sum for removing 2500 defects hence is 6000 hours.

In this example, the time saved if or not conducting reviews would be 14 000 programmer hours, or about seven years work for one software engineer.

However, a high yield does not guarantee saved time. In reviews there are two factors, at least, that affect time costs: yield percentage and time spent in defect removal activities. In this study design and code reviews introduced at Exercise 7 improved the students’ average yield significantly as Figure 5.2 shows.

![Figure 5.2: Average yield for PSP exercises.](image)

As discussed above a high yield does not automatically save time. It is interesting to consider the amount of time spent removing defects, in addition to examining the yield value. Hence, time spent in defect removal activities, with or without reviews, is compared in relation to the number of injected defects.

In PSP, there are two metrics that describe the Cost Of Quality, COQ, i.e. effort spent in defect removal activities:

**Failure COQ**

\[
\text{Failure COQ} = 100 \times \left( \frac{\text{compile time} + \text{test time}}{\text{total development time}} \right)
\]

**Appraisal COQ**

\[
\text{Appraisal COQ} = 100 \times \left( \frac{\text{design review time} + \text{code review time}}{\text{total development time}} \right)
\]
¼ \((\text{total development time})\)

To compare Failure + Appraisal COQ versus Failure COQ (before introduction of reviews) would only describe the difference of time spent in defect removal activities, with versus without reviews. A better way to compare how much more efficient, on a defect level, defect removal is with reviews is needed. This issue is addressed by comparing the total number of defects injected, with the total time spent in defect removal activities, with versus without conducting reviews. For Exercises 1-6, where no reviews are conducted, compile and test time are divided by the total number of defects injected. For Exercises 7-10, which involve review activities, the time spent in compile, test, design and code reviews are divided by the total number of defects injected. The number of minutes per defect better describes the effectiveness of reviews since it considers the number of injected defects and not only the time spent in reviews.

The results show that use of reviews requires 19.05 minutes per defect, while not applying reviews requires 24.36 minutes per defect. From Exercises 7 to 10 an average of 24.7 defects were injected. Hence, 19.05 multiplied with 24.7 turns into a total of 471 minutes and 24.36 multiplied with 24.7 is 602 minutes, as Figure 5.3 illustrates.

![Figure 5.3: Time spent in defect removal activities for Exercises 7-10, with versus without reviews.](image)

The introduction of design and code reviews in defect removal activities improved the metric Number of minutes per defect, by 21.8% or in actual saved time 131 minutes for completion of Exercises 7-10.

Despite the fact that the students did not update their checklists, and 43% of them had poor level of detail on the checklists, the introduction of reviews saved time. The students’ average of yield was 36%, when other
PSP research indicates an average of 70%.

5.6 Concluding Remarks

The level of detail of the checklist, The update of the checklist with a new defect profile and The number of reviewed LOC/Hour are criteria that affect the efficiency of a code review according to Humphrey [41]. Although the results from this study is not statistically significant, they indicate that update is the criterion with the most impact, followed by the criterion level of detail of checklist. The results from the study indicate that the criterion LOC/Hour has the least impact on code review efficiency. Further, the results show that conducting design and code reviews saved about 22% in defect removal time, compared to if defect removal activities only consisted of compile and test activities. The average removal time per defect with compile, test, and review activities was 19 minutes, whereas without reviews it was 24.

This study shows that it is possible to receive substantial benefits in software development, with relatively little effort. When an organization becomes aware of that it delivers software with many defects, it may address this problem in several ways. In many cases, according to our experience, these organizations start major SPI initiatives such as introducing CMM or RUP and so forth in order to improve the quality of the software development process. However, as stated by Johnson and Brodman [49] there are difficulties to overcome when embarking on these major SPI efforts. It is our experience that often these SPI initiatives do not prevail. One reason for this may be that organizations perceive that the amount of effort and time associated with such an SPI project is too much. Another reason may be that it usually takes considerable time to receive benefits from SPI efforts at an organizational level due to for instance a high communication factor. This lack of feedback tends to reduce the organization’s commitment to the SPI initiative, as argued by Kotter [56]. Instead, after some time the organization may continue to develop software as it did previously.

However, as this study shows substantial benefits may be gained with relatively small means as exemplified by the introduction of code reviews which reduced the defect removal time with over 20%. Further, the results of the study indicate that to improve the efficiency of a code review it is important to regularly update the code review checklist. Thus, an important conclusion from the study is that software development can be improved with small means, such as introducing code reviews as this study did. It is not necessary, and sometimes not even preferred, to address software development problems by embarking on major SPI initiatives. These initiatives may take considerable time and effort which an organization may not be ready for.
Chapter 6

STUDY 2: SPI SUPPORT BASED ON NON-TECHNICAL CRITERIA

6.1 Background and Problem

For organizations to be efficient when developing software it is important to support improvement in the roles that participate in a project. People should have high quality in their working skills. To improve this aspect, individuals active in software development should have means to improve their work processes. However, most software development approaches like Rational Unified Process, RUP [57] and the Team Software Process, TSP [43], focus on support for process improvement at an organizational or team level. Process improvement support at an individual level is not common in the software development community.

As software development is growing increasingly more complex new areas and roles are introduced that help contribute to the software development activity. Today, there exist roles that do not address traditional programming-related tasks such as design, code and test activities. Some of these roles work with so called soft factors. Chakrapani [19] provides one definition of a soft factor:

"Soft factors are used as a collective term for factors that are difficult to quantify exactly, i.e. non-technical aspects."

Following the reasoning in the statement made by DeMarco [24] "You cannot control what you cannot measure", it may be a challenging task to provide SPI support in a software development context to activities which
contain such non-technical criteria. This depends on the fact that these factors tend to be difficult to quantify by objective means. Soft factors cover a broad spectrum of activities and concepts for example including group dynamics and motivation of people. Although often difficult to quantify, there exists improvement support based on the application of non-technical criteria in areas such as project and change management, provided for instance by Lientz and Rea [60] and Johansson [48]. However, the underlying assumption for this study is that it is more difficult to provide SPI support when managing non-technical criteria in software development environments. This assumption is based on three factors. First, software development is a complex activity where teams often consist of people with specialized competence areas such as programming, testing, project management and so forth. This aspect can hinder communication as people may have difficulties of understanding each other, which makes it difficult to know what or how to improve a soft factor. Second, there is often a lot of changes in software development, not only in requirements but also for instance in team constellations and use of technology. This may have a negative impact when improving soft factors as it may become difficult to realize in which contexts or situations the support for soft factors should be provided for. Third, software development projects are often under severe time pressure. Thus, the collection and analysis of qualitative data, which is not automatic, requires time which may not be affordable. These factors or characteristics of software development is mentioned by Brooks [15] where he argues that progress in software engineering will not be substantial due to among other things the complexity and changeability inherent in software development which may create tight schedules, misunderstandings and disorientation how to develop software.

To investigate to which extent the application of non-technical criteria can provide SPI support in a software development context, support was developed for a role in software development who worked with soft factors. The reason for developing support for managing soft factors to a role was based on the relative success with PSP. PSP is a process at an individual level which provides support for the software engineer role. Although it has been difficult to transfer PSP into industrial usage, as mentioned by Humphrey [44], there has been a number of studies for instance by Ferguson et al. [30] which show that PSP may be an efficient way to develop software. This motivates development of support for other roles, which may help improve the individual performances when applying the roles. The BPA role was chosen to receive process improvement support since it is a role who to a large extent works with soft factors in software development and it had not, prior to this study, received any process improvement support. The BPA includes tasks that are quantifiable, but the main part of the tasks concern soft factors. The BPA role is important since it identifies business requirements and transforms them into software development requirements.
Further, it is a communication channel between the organization and the software developers which helps develop software that is meaningful to the organization. A more complete description of the BPA’s role in software development projects is provided in Section 6.3. There exist many roles that handle soft factors, and sometimes they share common work tasks. For instance, as stated by Gilbert [35] there may not be much difference between being a project manager and a BPA. Today, project managers may be expected to assume the role of the BPA. The BPA role is a collection of skills, and can be applied by more than one person. However, it is not really important to distinguish what separates these roles. The purpose of the research is to investigate the meaningfulness of developing SPI support based on the application of non-technical criteria at an individual level in software development. Thus, although the BPA role was chosen to illustrate how SPI support can be provided by managing non-technical criteria in software development, other roles that handle similar soft factors may benefit from using the framework. The output from this work is twofold. One part of the contribution is the contents of the framework which is accessible at our web site\textsuperscript{1}. Hence, interested readers are encouraged to apply the metrics, guidelines and templates in their daily work. A more detailed presentation of the framework is provided in Section 6.5. The second part of the contribution is an answer to the question whether it is meaningful to develop SPI support based on the application of non-technical criteria in a software development environment.

The outline of the study is as follows. Section 6.2 contains related work and Section 6.3 includes a description of the BPA. Section 6.4 presents the research methodology. Section 6.5 provides an overview of the framework. Section 6.6 presents the research results and Section 6.7 presents the conclusions from the study.

### 6.2 Related Work

An important but often overlooked issue in software development is the role soft factors play in software development and how these factors may be controlled. This view is shared by McConnell [63] where he states that although programmers have the same amount of experience, can their productivity differ tenfold. This can of course partly be explained by different use of applications and so forth which address the technical side of software development. However, as argued by McConnell it is the human-oriented factors that have the largest impact on productivity and quality of the delivered software. McConnell identifies motivation and the availability of senior staff as the main factors that contribute most to enhanced software development performance. The ability to motivate personnel leads to even more commit-

\textsuperscript{1}http://www.dsv.su.se/~haralds/
STUDY 2: SPI SUPPORT BASED ON NON-

ted people as they tend to recognize their own increased performance which makes them eager to perform better in future projects. Further, senior staff provides a substantial amount of experience (both technical and social) to a software development project which has a positive effect on the rest of the project participants.

Wohlin et al. [98] present an approach how to control soft factors in order to reduce the time to market for a software product. The approach is based on analyzing 12 projects, where 10 soft factors have been graded in each project on a scale from 1 to 5 where the hypothesis is that high values result in fast projects. By analyzing the data, correlations between the soft factors and the completion of a project (i.e. the time to market) could be identified. Further, Wohlin et al. emphasize that the understanding and knowledge of soft factors help improve planning and control of software development projects. The study shows that controlling soft factors may improve specific activities in software development, such as planning and delivery of software as addressed in the study.

As argued above, soft factors play an important part in software development. However, another important aspect worth considering is the collection and analysis of data during software development. Constant feedback enables identification of weak areas in the software development process, which if addressed, may improve the performance. An SPI initiative follows more or less the steps advocated in the Plan Do Check Act, PDCA, cycle for instance described by Tague [94] and Masaaki [61]. The main philosophy behind this cycle is to plan what to do, then do it. After that an assessment is made whether the introduced changes actually achieved the expected improvement or not. The final step is to decide whether to implement the changes or to start all over again. The PDCA cycle facilitates continuous improvement as the cycle is never ending, in the sense that the output of a completed cycle can be the input of a new cycle. The PDCA cycle is not specifically aimed to be used in software development but is generic in its applicability. A model similar to the PDCA cycle is the Quality Improvement Paradigm, QIP, developed by Basili [9]. The QIP model facilitates continuous improvement and in contrast to the PDCA cycle it was developed to be used in software development. It applies six steps when implementing SPI initiatives. They basically follow the four steps in the PDCA cycle. However, the QIP model differs from the PDCA cycle in the sense that it has a stronger focus on learning. The QIP model emphasizes that each project and environment is unique and therefore requires unique ways to implement SPI initiatives in. The experiences and results from these SPI initiatives are stored for future usage.
6.3 Business Process Analyst

In software development projects, the BPA has a wide variety of responsibilities. They may include translating business requirements into software requirements, developing training material for software tools or specifying and executing test plans in collaboration with involved parties in an organization. The BPA is a multifaceted role, that is its work issues range from areas such as motivating people to modelling business processes. Thus, not everything the BPA does is related to software development. For instance, the BPA should be familiar with techniques for organizational design, process improvement, technology assimilation, organizational change and process modelling. The role should identify opportunities for improving business processes, organizational design and corporate culture. Thus, many of its tasks are not related to software development activities. However, the purpose of the SPI support is to provide means for the BPA role to improve software-related tasks in a software development context that concern soft factors.

A BPA involved in software development is defined by Kruchten [57] as:

"The business-process analyst leads and coordinates business use-case modelling, by outlining and delimiting the organization being modelled. For example, establishing what business actors and business use cases exist and how they interact."

This definition only describes high level goals for the BPA. For instance, it mentions that the BPA must be able to work in groups of people and lead teams, but leaves it at that. To define an SPI framework which provides meaningful support to the BPA’s role in software development a more complete and detailed picture of the BPA’s working situation and personal qualities should be obtained. This detailed picture was obtained through interviews conducted with BPAs from the business world. The result from these interviews was a description of the BPA’s work areas in software development. The work areas were identified as Analysis and Modelling, Motivation of People, Team Work and Process Improvement. Support was developed for the BPA role to improve its work in these areas.

6.4 Research Methodology

This study investigates the meaningfulness to develop SPI support based on application of non-technical criteria in a software development context. The research question is formulated as:

*To which extent can the application of non-technical criteria provide SPI support in a software development context?*
The support was provided in form of an SPI framework for the BPA role in software development. The research approach for developing the BPA framework is divided into three steps. First, information is gathered to understand work goals and needs of the role receiving the process improvement support, whether it may be a software engineer, BPA or any other role involved in software development projects. This is necessary in order to develop templates, metrics and so forth that address important and meaningful process improvement aspects of the role. After gathering information about the BPA role in software development the contents of the framework, i.e. the actual SPI support, was developed. The approach used was to read literature on managing non-technical criteria by for instance Lientz and Rea [60], Johansson [48] and Chakrapani [19] and to adapt appropriate material to support the BPA in software development. Further, some of the support were developed without prior references to existing templates or metrics. The third step concerns validating the framework, for instance through assessments which was used in this study. The validation step provides an insight of the level of support from the framework by identifying which parts of the framework that are regarded as useful and which are not. This may be an iterative process where the end product may be a modified and updated framework that better supports the role in question. Another important part of the validation activity is to evaluate the framework in practical use. Here, new insights may be gained how the framework should be used or modified to better support the BPA role. The work presented in this study addresses all three, hence the gathering of information, development and validation of the BPA framework. However, the validity aspect which involves industrial use was out of scope of this study as it requires a lot of time and effort which was not available. As a consequence, this activity belongs to future research.

Below, a more detailed presentation of the activities that were taken to develop and validate the framework (i.e. the three steps described above) is provided. The study consisted of the following main steps. First, a number of BPAs from the business world were interviewed to realize important goals in their work. After determining these goals a literature survey was conducted on related works before defining the framework contents, where interrelations between the framework contents and the identified goals were made. The BPA framework was then assessed by BPAs from the business world. Finally, the BPAs were interviewed. The interviews provided more information regarding the meaningfulness of a process improvement framework for the BPA, involved in software development. After evaluating the results an assessment of the framework’s meaningfulness was made.
6.4. Selection of Study Participants

Due to time and budget constraints, no sophisticated random sampling technique was applied when choosing BPAs to interview. Instead, available BPAs were interviewed in the local region. This approach could be summarized as convenience sampling described by Wohlin et al. in [97]. To ensure that the chosen BPAs were representative of the target population, they were subjectively assessed through interviews before participating in the study. Although the study participants were few (seven in total) they had different backgrounds, experiences and competences which indicate a wide variety in the sample data thus increasing the likelihood that the sample data is representative of the target population.

6.4.2 Developing the SPI Support

Below, the main steps are described concerning the procedure to develop support for managing non-technical criteria at an individual level in a software development context.

The First Round of Interviews. The purpose of the first round of interviews was to gain information from BPAs in the business world, and identify important goals in their work. The interviews were prepared by studying RUP’s definition of a BPA. This way, the questions were based on relevant material. The interviews were of the type open-guided interviews described by Lantz in [58]. Open-guided interviews mean a wide research question, where the interviewees can respond with open answers. The questions were starting points for discussions, as the author of this thesis did not know what goals were important for a BPA involved in software development.

Developing the Framework. The first round of interviews provided work goals for the BPA. Based on these goals contents of the framework, i.e. guidelines, metrics and templates were defined. The Goal Question Metric, GQM, technique, for instance described by Berghout and Solingen [91] was applied when developing the framework. For instance, one goal was to make people motivated to participate in business improvement work. To reach this goal it is important to know the current status of the team members’ motivation, thus the question How motivated are the team members? was constructed. To address this question two metrics were defined. The metric Commitment Indicator is a subjective metric that is defined as the level of commitment of a team member for a certain time period, decided by the BPA. The metric Complaint Metric is defined as the number of received complaints from a team member for a certain time period, of how the BPA is leading the change analysis work. The contents of the framework and
their interactions were graphically displayed, to better understand the couplings in the framework.

**The Assessments.** The study participants assessed the framework based on its usefulness for supporting the BPA in a software development context. The assessments were based on how the BPAs perceived the use of the framework would help them in software development, as they had not applied it in real situations. A questionnaire was used in the assessments. Each assessment was graded with five alternatives. The ordinal and absolute scale types were used on the questionnaire, because in the ordinal scale type an ordering among classes exists. Further, the absolute scale was chosen since it was important to count how many times each alternative was chosen.

**The Second Round of Interviews.** The purpose of the second round of interviews, was to gain additional information about the framework’s support to a BPA involved in software development. The assessments provided raw data. The interviews provided more information that the study participants could not express in the assessments. The questions for the second round of interviews were based on the assessments. That way, a deeper understanding of the BPAs perception of how the framework would support them in their work was gained, as it was possible to follow up on issues encountered from the results of the assessments.

### 6.4.3 Validity

It was the same people that were interviewed as did the assessments. Thus, it is possible that a different result would have been obtained if it was not the same BPAs who were interviewed as did the assessments. However, due to time and economical constraints this option was not feasible.

Further, the study participants were only seven in total which may affect the validity of the results. Although this is not a quantitative study where it is important to have a lot of data, a larger number of study participants would have increased the likelihood that the sample was representative of the population.

As mentioned by Lantz [58], it is important to reflect the opinions of the study participants in qualitative research. Thus, to gain a better understanding of how the BPAs perceived the support from the framework, a second round of interviews was conducted to complement the results from the assessments.
6.5 Framework Overview

The purpose of the framework is to help the BPA improve his/her work routines in software development. The framework helps the BPA focus on important aspects in software development that relate to the areas the BPA is involved in. Further, the framework provides means for collecting and analyzing process data regarding the BPA's performance which help realize opportunities for improvement. For instance, the framework helps the BPA improve his/her modelling activity by providing a questionnaire which contains questions that address important modelling aspects to consider in software development organizations. As stated above, the SPI support is provided in form of a framework. An alternative would have been to provide the support in form of guidelines, but a limitation with guidelines is that they only provide suggestions without really stating when to apply these suggestions. Further, guidelines are no means for collecting and analyzing data which is a vital aspect in SPI. These limitations do not apply to an SPI framework. A framework provides an environment defined for a purpose which supports activities in that environment. Thus, a framework may include both guidelines and instructions when to apply them and means for collecting and analyzing data. While the underlying principles of the framework such as guidelines for improving team work and so forth are useful, the use of a framework provides more support. The framework provides the BPA with means for improving software development efforts by answering the questions what, why, when and how to apply the process improvement support. These questions are not answered when only applying the underlying principles. The framework consists of both unique material and support from other sources which have been adapted to fit the BPA in a software development context.

The defined framework consists of guidelines, metrics and templates. Table 6.1 presents the areas which the framework provides support for, and the number of guidelines, metrics and templates in each area.

<table>
<thead>
<tr>
<th>Area</th>
<th>Guidelines</th>
<th>Metrics</th>
<th>Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis and Modelling</td>
<td>14</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Motivation of People</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Team Work</td>
<td>30</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Process Improvement</td>
<td>12</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>

The framework contains the same types of elements as are present in the PSP framework. Results from applying PSP have shown that the software engineer is supported with means to improve his/her development process. Thus, the BPA framework is developed on a solid structure as it uses the
same types of elements as PSP does.

The framework is based on literature such as Lientz and Rea [60] on measuring and managing non-technical criteria. The material was reviewed from a software development point of view. The chosen material was then adapted to support the BPA in software development. The intention of the BPA framework is to assist the BPA realize his/her work progress, analyze and model business processes, determine the impact of process changes and lead a team working with improving business processes. Information regarding the framework elements’ applicability in their contexts is provided below. For each area, some of the framework elements have been used as examples when describing how and why the framework elements should be applied in their respective contexts.

**Analysis and Modelling.** This work area contains a *model review checklist* which helps the BPA to verify that a model covers relevant aspects when modelling. *The guidelines* contain advice when analyzing or modelling, such as choosing suitable notation when modelling. A metric used in this work area is the *cyclomatic complexity* described for instance by Fenton and Pfleeger [29]. The metric is used to investigate the number of independent paths in a graph to make sure no paths are forgotten in path testing. However, the metric was applied on business processes and not on source code as it was initially defined for.

**Motivation of People.** This work area includes a *motivation assessment template* which helps the BPA understand how his/her behavior affects the motivation of the rest of the team. The team members assess the BPA on a number of factors that affect their motivation. *The guidelines* contain advice how to motivate people, such as creating a vision for the team to work towards or making sure that the software project has enough resources so that the team perceive their work as important. A metric for motivating people is the *commitment indicator*, which is a subjective metric that is defined as the level of commitment from a team member under a time period as perceived by the BPA.

**Team Work.** The *issue resolution chart* helps the BPA to monitor resolved and unfinished issues. The *issue management form* is a way for the BPA to define the characteristics of an issue such as who is assigned the issue, when it was solved and how. This provides control and structure when dealing with soft factors which often are vaguely described and complex. *The guidelines* concern issues how the BPA should act to maximize the effects from team work. One metric is the *issue elimination ratio*, which measures number of completed issues per time period divided with number of reported issues per time
period. A high value indicates that the team is efficient in solving issues.

**Process Improvement.** This work area includes *the task and schedule planning templates.* These templates help the BPA to keep track of how actual task and time schedules relate to planned schedules. The templates offer a good overview when prioritizing between tasks, which is important as changes are common in software development. *The guidelines* cover issues such as advice on establishing peer networks. One metric to be applied in the process improvement work area is *the defect ratio.* It is defined as the number of injected defects in the project, divided with the average number of injected defects from earlier projects. A high value indicates low work quality in the current project. The meaning of a defect is defined by the BPA.

The intention with the framework is to support the BPA to improve his/her work in a software development context. In more concrete terms, this is achieved when the BPA uses the framework elements (e.g. guidelines, metrics and templates). For example, before the BPA will run a meeting the BPA may review appropriate guidelines how to properly plan and conduct the meeting. Another example of support is the possibility to (subjectively) measure team members’ commitment to a project. This information is useful as it helps the BPA to realize if some of the people he/she is working with tend to lose interest in their work. Thus, by using appropriate elements of the framework in different situations the BPA role is supported to improve its work in a software development context.

The BPA works with many people and in different contexts which change the purpose and goal of his/her work regularly. This advocates use of different parts of the framework depending on the actual situation the BPA is involved in. This condition does not suit the use of maturity levels used for instance by PSP where parts of the framework are introduced gradually in a predetermined way, since it is not possible to know when the BPA needs certain SPI support. Instead, the structure of the framework is based on the four areas mentioned previously in this chapter where each area consists of a number of goals which are important for the BPA to fulfill in order to be successful in software development projects. The framework provides an overview of the four areas and their associate goals, where the BPA can use parts of the framework as appropriate depending on the current needs. This approach may be classified as a context-dependent framework structure where the initiative to choose framework elements is transferred to the user, instead of the contrary as with PSP. The structure (i.e. areas and goals) of the framework is presented in Figure 6.1.

The framework is provided at our web site\(^2\), as mentioned in Section

\(^2\)http://www.dsv.su.se/~haralds/
6.1. However, although the maturity level approach was not used the notion of gradually introducing the framework elements still applies as the BPA only uses those elements that are valid for the actual situation. In the BPA framework, guidelines outnumber metrics. The reason for this may be that soft factors are hard to develop metrics for, due to their abstract nature. PSP, which supports the software engineer in design, coding and testing has equal number of guidelines and metrics. Thus, it seems easier to develop meaningful metrics for tasks that do not concern soft issues.

6.6 Results

In addition to the developed contents of the framework the results also contain an assessment from the study participants, regarding how they perceived that the framework would help improve their software development activities. The average grade on the framework’s applicability consists of their assessments of the framework and answers from the second round of interviews. The results from the assessments are presented in Section 6.6.1, and the results from the interviews are presented in Section 6.6.2. Both results help answer to which extent the application of non-technical criteria can provide SPI support in a software development context.

6.6.1 Assessment Results

The BPAs assessed the framework’s support for the BPA to fulfill identified goals in the work areas. Figure 6.2 presents the assessment results. It presents an aggregated view where the columns refer to the work areas. A value of five indicates maximum support from the framework. A value of one indicates minimal support.
6.6. RESULTS

Figure 6.2: The assessment results.

Column one refers to the work area Analysis and Modelling. Column two refers to the work area Motivation of People. Column three concerns the work area Team Work. Column four refers to the work area Process Improvement.

The total average value from the BPAs was 3.3 out of 5 regarding the framework’s support for a BPA involved in software development. Thus, the BPAs perceived that the framework would probably help them perform better in software development projects. The goals that received the lowest grades were goals in the area Analysis and Modelling, and the goal Run Efficient Meetings in the area Team Work. The goals that received the highest grades were goals in the area Motivation of People and the goal Constantly improving the work process of the group in the area Process Improvement. As Figure 6.2 shows, the BPAs assessed that the work areas received similar amount of process improvement support.

6.6.2 Interview Results

A summary of the answers is presented below to provide an understanding how the BPAs perceived the framework. The first two questions concern improvement support from the existing framework. The last two questions concern unattended aspects that the BPAs perceived would improve the framework’s level of support.

Are there some elements that you would apply in your work?

The BPAs perceived that they would apply the main part of the framework’s contents, especially the elements in the work areas Team Work and
Process Improvement. They were deemed as particularly useful as they addressed project progress issues and continuous improvement of team members.

**Are there some elements that you would not apply in your work?**

The BPAs perceived that the elements in the work area Analysis and Modelling were not so useful as they assume use of formal models, whereas the BPAs many times work with informal models where it is more important to communicate ideas and concepts than to develop consistent formal models.

The next two questions address unattended aspects that the BPAs perceived would improve the framework’s level of support.

**The average assessment grade was 3.3 out of 5. What do you think it would have taken to raise the value to 4-5?**

The BPAs were of the opinion that to increase the level of support the framework should provide support when dealing with political forces in an organization.

**Are there some work areas or goals of the BPA that have not received process improvement support?**

The BPAs were of the opinion that a BPA needs to understand an organization’s vision or view on future actions. That issue is not addressed by the framework.

### 6.7 Concluding Remarks

The result from this study indicate that the application of non-technical criteria can provide adequate SPI support in a software development context. The BPAs’ overall assessment of the framework’s support was 3.3 out of 5. Thus, they perceived that the BPA framework would likely help them perform better in software development projects. The interviews made it clear that the framework in the work area Analysis and Modelling focused too much on a system approach, whereas it should focus more on analyzing and modelling an organization, i.e. a social phenomenon. Regarding the work area Motivation of People, the BPAs appreciated the templates for evaluating the team’s relation to the BPA and the guidelines identified important motivation aspects. The BPAs perceived that an experienced BPA would not likely achieve major improvements in his/her work as a new BPA probably would, since it provides support to several work areas of the BPA which he/she may be unfamiliar with. This conclusion is based on opinions
6.7. CONCLUDING REMARKS

from the interviewed BPAs. An important contribution from the interviews was an improved understanding of what parts of the framework that provide versus do not provide process improvement support. Further, another insight gained from the interviews was that the framework should support the BPA to manage political issues in an organization as this is seen as an important part of the BPA’s work, by the interviewed BPAs. The next step after addressing these issues, would be to transition the framework into industrial use to better understand difficulties and problems encountered when applying the framework.

The results from this study indicate that the application of non-technical criteria can provide adequate SPI support in a software development context. A concern though mentioned in the second round of interviews is that the BPA is often involved in many activities and may not find the necessary time to collect and analyze process data, in order to identify and implement improvement initiatives. Further, soft factors concern non-technical aspects such as trust and commitment to a project which tend to be difficult to measure, partly because it may be problematic to define a meaningful metric but also because the collection of data is not automatic. These two aspects show that time is a vital aspect when developing SPI support based on the application of non-technical criteria in a software development context. This factor was one of the reasons for conducting this study, as the assumption was that time pressure would hinder the support to soft issues. This assumption is supported by the study results. Hence, tight time schedules in software projects seem to have a negative impact managing when managing non-technical criteria in software development contexts.

Another result from the study was that people may change positions and roles during a software development project. This may have a negative impact on the management of non-technical criteria since for a soft factor such as trust it is important to establish a dialogue with the other person, and if that person leaves it becomes impossible to continue the dialogue. The third aspect that may have an impact on the management of non-technical criteria is the fact that software development is a complex activity, where teams often consist of people with specialized competence areas. This fact may complicate the application of non-technical criteria since people may have difficulties in understanding each other. Although this aspect was not confirmed in this study it may still be valid which future studies in the area may confirm.
Chapter 7

STUDY 3: INDIVIDUAL SPI SUPPORT IN RUP

7.1 Background and Problem

There exist a lot of software development frameworks today, e.g. Capability Maturity Model [73], Team Software Process [43] and ISO 9000 [72]. The frameworks differ in definition and are based on different general models or paradigms of software development, e.g. the waterfall model, formal transformation and evolutionary development. Unfortunately, these frameworks are not complete in the sense that none of them offers complete support to software development. Some frameworks are better suited in a specific context, whereas others suit better in other contexts. It is important that these frameworks evolve, based on experience from conducted projects and new technology so that they offer best possible process support. Most software development frameworks provide support for development of software by teams or organizations, e.g. RUP [57] and [46], CMM [73], TSP [43] and the IDEAL model [1] as confirmed by Zhong et al. [100]. However, many of these frameworks do not support processes at an individual level. The focus and effort of coordinating work activities at a team or organizational level seem to leave out support for the workers on how to monitor and control their software processes. As a consequence, it is more or less up to the individual worker himself to find out ways of improving his/her software development process.

RUP is a software development process that supports activities in software projects conducted by teams or organizations. RUP defines process elements such as workers, activities and artifacts that when combined properly may help an organization to conduct a software project in an efficient manner. However, the process itself does not provide any magical solutions. It is the workers, i.e. actual human beings, that are the driving force of a project. Without the project participants’ work, no process or tool in
the world would alone finish a software project. The end result, e.g. time costs or the quality of a released system and so on, depends on the quality of the project participants' software processes. RUP provides limited support for process improvement issues to its workers. Instead, it provides a framework for how, what, when, and by whom various activities should take place in order to get a satisfying result. In other words, an organization using RUP with competent and experienced people, will do better compared to an organization consisting of incompetent and inexperienced employees. It is important, especially for organizations with inexperienced people, to quickly improve their employees’ skills in order to save time and money. RUP does not provide an answer to this issue. Thus, the research question is formulated as:

*How can SPI support at an individual level be introduced into RUP?*

PSP provides SPI support at individual level. Keeping the research results of applying PSP in mind (e.g. [41], [93] and [71]) it can be concluded that a similar process improvement awareness in RUP would help its workers to improve their processes, and thus improving the result of the organization as a whole. However, RUP’s workers would find it hard to apply PSP material unmodified since RUP and PSP differ significantly in contents and purpose. PSP material should be modified so that it suits the RUP context. If every RUP project was conducted in the same manner, i.e. all artifacts, activities, work flows, life cycle models and so on never changed, PSP material would only need to be modified once. However, this is not the case. Every project is unique and is conducted in a certain manner. This means that RUP material, e.g. activities, artifacts and workers should be adapted to the work situation at hand. Therefore, this work presents how PSP at an abstract level is incorporated into RUP. This way, every RUP project can adapt PSP material to its own needs, and apply PSP concepts accordingly. Further, the work presents expected benefits of incorporating PSP into RUP.

A question to ask is why RUP was chosen to receive PSP material? There exist other SPI programs, e.g. SPICE and CMM; why RUP? RUP was chosen to receive PSP material due to three major reasons:

1. RUP has clearly defined workers, i.e. roles, who are responsible for developing artifacts and carry out activities. Having this in mind, there exist good preconditions to incorporate PSP into RUP since PSP is an SPI program at an individual level and can be applied by RUP’s workers.

2. RUP provides little support for its workers how to improve their software processes. As a consequence, PSP can provide support to RUP’s workers regarding process improvement.
3. RUP and PSP share common notions how to conduct software development and semantically equivalent definitions of software development terms. These aspects promote a merge of the processes since they simplify the procedure of incorporating one process into the other.

Related work is provided in section 7.2. RUP is described in Section 7.3. Section 7.4 presents differences and similarities between RUP and PSP. In Section 7.5, PSP is incorporated into RUP. Section 7.6 contains concluding remarks.

7.2 Related Work

Capra et al. [17] and Escala and Morisio [28] describe how PSP was modified to be applied on a team level. The company, an Italian medium sized enterprize with about 160 employees, worked with corrective and evolutive maintenance on a product. The company needed a process for improving work activities on a team level. Some activities and measures from PSP were adapted accordingly. For instance, individual estimations were combined to produce a team estimate and plan for the project. The developed metrics provided information of the team’s performance. Further, PSP elements were adapted to the company’s working environment, i.e. corrective maintenance of software products. To record the changes in the process model, the company developed an action inheritance tree that described how different actions related to each other.

Watts Humphrey, the founder of PSP [2], has addressed the issue of process improvement support for teams developing software. He has developed TSP [43] which addresses these issues. TSP provides guidance of how to control, monitor and improve software development in an industrial team environment. Like RUP, TSP is a process for software development by several people. However, TSP focuses on smaller teams whereas RUP is used in larger software development projects. TSP has different roles in a team but unlike RUP, role practice is combined with ordinary software engineering tasks.

7.3 Rational Unified Process

The Rational Unified Process\(^1\) [46], RUP, is a software development process which supports software project activities from analyzing a business case to the transition of a developed system into industrial use. RUP is a software development process at an organizational level. RUP supports software

\(^1\)In RUP, a worker refers to a role in a project. A person can have several roles, i.e. act as several workers. A worker is not a human being, it is a role a human being can take on.
RUP is characterized by four key aspects: Use-Case Driven, Architecture-Centric, Iterative and Incremental.

By stating that RUP is use-case driven means that the process applies use-cases as a way of describing the interaction between users and systems. By elaborating with the interaction through use-cases, requirements can be captured which could otherwise be difficult to find. Architecture-centric means that RUP describes in different views important static and dynamic aspects of a system before it is implemented. This way, software engineers are in a good position to produce an architecture that may withstand the winds of change, as new requirements likely will evolve that imply other design solutions. Iterative development applies a "Divide and Conquer" paradigm. By designing, implementing and testing small parts of a project specification, misinterpretations, risks and defects can be realized early in a project’s life cycle. This way, the system will likely evolve gradually in the right direction. RUP is incremental in the sense that each iteration is a "mini-project" of its own with a defined output, an increment. The increments are subgoals that contribute something to the evolvement of a system, e.g. adding functionality or improving reliability. In order for RUP to implement the key aspects above terms like cycles, phases, work flows, risk mitigation, quality control, project management and configuration control are applied in the process. This makes RUP an integrated process, covering a wide range of aspects concerning development of software systems in organizations. RUP has a number of workers and each worker is responsible for a certain work area in software development. An employee can act as several workers depending on his/her area of expertise. A worker performs activities which is something a worker does that provides meaningful results to a project. Activities have input and output artifacts. An artifact is a work product of the process, e.g. a software development plan, test guidelines and so on. Artifacts help workers know what to do in activities and can provide additional information for decisions regarding project-related issues.

RUP has been applied by a number of enterprises developing software-intensive systems as described by Devlin and Royce [26]. They state some aspects of RUP that make RUP successful to use:

- The ability to rapidly develop complex software products.
- The ability to respond to changing customer requirements.
- The ability to take advantage of new business opportunities by providing an adaptable process.

RUP applies a modelling language, UML, which stands for Unified Modelling Language. It provides a standard communication language for software
engineers to visualize their ideas, so that people involved in a RUP project can understand presented ideas. The UML has a graphical notation and well defined semantics. Further, RUP is documented online on a HTML format. This provides easy access to the process.

7.4 Comparing RUP and PSP

This section presents differences and similarities between RUP and PSP. This is necessary in order to better understand why and in what way PSP is incorporated into RUP. The presentation is divided into two subsections. Section 7.4.1 focuses on application of the processes, i.e. how the processes support software engineering activities and how they are defined. This part of the comparison is needed to motivate changes made to PSP in order for it to fit into RUP. Section 7.4.2 presents terminologies used by RUP and PSP. This part of the comparison is needed to understand differences and similarities in RUP’s and PSP’s elements, or artifacts to use the RUP term.

7.4.1 Application of the Processes

This section presents how RUP and PSP support software engineering activities. First, the issue in question is presented, followed by RUP’s and PSP’s definitions. This is followed by a comment on how RUP and PSP differ and are alike concerning the issue at hand. Table 7.1 to 7.6 show RUP’s and PSP’s definitions regarding software development issues.

7.4.2 Terminology of the Processes

RUP and PSP have several software terms in common. However, they sometimes label software terms that are semantically equivalent, differently. Further, some software terms have the same labels but different semantics. To perform a successful incorporation of PSP into RUP, it is vital to know which terms in PSP and RUP that relate to each other to understand where in RUP the PSP material is appropriate to introduce. To address this issue, different software terms with the above explained characteristics were listed and explained, with PSP and RUP definitions. First, terms with the same label but with different semantics are presented. Then terms semantically equivalent, but with different labels are explained.

A Comparison of Terms with Same Labels but Different Semantics. In this section, a number of terms used in PSP and RUP are explained. The terms used in the processes have same labels but different semantics.
Table 7.1: Comparing PSP and RUP.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Development Life Cycle</strong></td>
<td>Concerns areas in software engineering that a process covers, and the way the process is applied.</td>
</tr>
<tr>
<td>RUP definition</td>
<td>RUP covers many areas in software development. It addresses issues from requirements management to transition of software to customers. RUP provides a thorough process support for conducting large software projects. RUP applies an iterative life cycle.</td>
</tr>
<tr>
<td>PSP definition</td>
<td>PSP focuses on traditional software development issues such as design, code, and test activities. The process offers only guidelines in areas such as requirements elicitation and no support at all regarding, for example, transition of software systems to customers. The first six levels in PSP apply the waterfall life cycle, whereas the last level to a larger extent applies iterative development.</td>
</tr>
<tr>
<td>Comments</td>
<td>RUP covers more areas in software development than PSP. However, this should not be considered a drawback for PSP. A process’ contents should suit the purpose of the process. In PSP’s case, the person who applies the process is often a software engineer involved in a bigger project, where issues such as transitions of systems to customers is not any of his/her tasks. RUP is iterative and incremental in its development process, which also the last maturity level in PSP is.</td>
</tr>
</tbody>
</table>

**PSP definition** A process where the software engineer examines his/her own products. The review is based on the software engineer’s defect profile.

**RUP definition** A group activity carried out to discover potential defects and to assess the quality of a set of artifacts. The review is based on recognized common defects.

**Comments** RUP also states that a worker writing code should carefully review it by himself before giving the code to a review group. However,
7.4. COMPARING RUP AND PSP

Table 7.2: A comparison between PSP and RUP regarding iteration assessment.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| **Iteration Assessment** | **RUP definition**<br>Addresses a process’ way of assessing an iteration in the development life cycle.<br>RUP partitions its development process in time periods called iterations. A short iteration is considered to be about one month in time. RUP stresses the need to assess artifacts and activities during and after an iteration, as stated in [57] ”Within the iteration, progress and risk must continue to be assessed (if informally) to ensure that difficulties do not derail the project”.
| **PSP definition**<br>PSP partitions the development process in short iterative time periods, called cycles. In PSP, the software engineer is recommended to keep each cycle to about 100 LOC of new and changed code. This usually means a cycle time of a couple of hours. After each completed cycle, the software engineer should review his/her performance and make any necessary changes to the software development process in order to improve it. Both PSP and RUP assess short periods of work. After an assessment, actions are taken to correct any areas in the process that need attention. This way, potential risks are detected early in the life cycle of a project and thus lesser damage, in terms of time, are received. However, PSP has more frequent iteration assessments since an iteration in PSP is much shorter than in RUP. However, RUP encourages its workers to assess their work on a frequent basis, even though no formal assessments are taking place. |
| **Comments** | the review does not use a checklist based on the engineer’s defect profile as is the case with PSP. |

**Phase**

**PSP definition** A process element with a definition and a structure.
### Table 7.3: A comparison between PSP and RUP regarding process tool support.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Tool Support</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RUP definition</strong></td>
<td>Degree of tool support in a process. RUP has tool support that covers all areas in its software process. In fact, RUP was designed to interact with tools that would support the process. As a consequence, RUP was developed at the same time as its tools.</td>
</tr>
<tr>
<td><strong>PSP definition</strong></td>
<td>PSP was not released with any tools, nor are there any references in the PSP literature about any software applications that offer process support. There exist some PSP applications developed by private persons. However, the tools lack support for sophisticated process support that would enhance the improvement of the software development process.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>Both processes stress the fact that software tools will make their processes more efficient and easy to use. RUP has tools integrated into its process. PSP has not. However, it emphasizes the need to have tools for process support in areas such as collecting and logging process data as stated by Humphrey [41].</td>
</tr>
</tbody>
</table>

**RUP definition** Time between two major project milestones, during which a set of objectives is met, artifacts are completed and decisions are made to move or not move into the next phase.

**Comments** PSP uses the word phase both when referring to small and large software development activities. RUP refers to effort spent reaching a major milestone.

**Cycle**

**PSP definition** A PSP2.1 process that produces a part of a product.

**RUP definition** One complete pass through the four phases: Inception, Elaboration, Construction, and Transition. The span of time between the beginning of the inception phase and the end of the transition phase.
Table 7.4: A comparison between PSP and RUP regarding collection and logging of process data.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collection and Logging of Process Data</strong></td>
<td>The process of recording and storing process data.</td>
</tr>
<tr>
<td><strong>RUP definition</strong></td>
<td>RUP offers vague advice about appropriate templates and metrics for gathering and logging of process data. This is a consequence of RUP being general in its definition, allowing a great variation of organizations to apply the process.</td>
</tr>
<tr>
<td><strong>PSP definition</strong></td>
<td>PSP provides predefined templates and metrics for logging time, size and defects. The templates and metrics support the software engineer to log process data.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>PSP has predefined metrics and templates, in RUP they must be defined and/or selected by the project participants.</td>
</tr>
</tbody>
</table>

Table 7.5: A comparison between PSP and RUP regarding number of involved people in the software development process.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Involved People in the Process</strong></td>
<td>Number of persons involved in applying the process.</td>
</tr>
<tr>
<td><strong>RUP definition</strong></td>
<td>RUP is a process that involves a number of workers, each worker responsible for a certain work area in the software development process. RUP is applied by organizations involved in large software projects.</td>
</tr>
<tr>
<td><strong>PSP definition</strong></td>
<td>PSP is a defined discipline for software process improvement, at an individual level. All the software development tasks are carried out by one person, the software engineer.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td><strong>Comments</strong> A cycle in PSP is a very short iteration, expressed in RUP terms. A PSP cycle lasts about a couple of hours.</td>
</tr>
</tbody>
</table>

**Baseline**

**PSP definition** An established process with basic measurements and a reporting format.
Table 7.6: A comparison between PSP and RUP regarding process maturity.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Maturity</strong></td>
<td>Addresses the issue of how much the process supports improvement of software engineering skills among its users.</td>
</tr>
<tr>
<td><strong>RUP definition</strong></td>
<td>RUP has one level of maturity. It offers some support to process improvement issues, at an organizational level. However, it provides only limited support to process improvement at an individual level.</td>
</tr>
<tr>
<td><strong>PSP definition</strong></td>
<td>PSP has seven different maturity levels, where the first level is the most basic one, whereas the last level contains more metrics and templates, thus a more advanced process. PSP provides process improvement support at an individual level.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>Both processes address the need to improve a software process. PSP provides explicitly a &quot;maturity ladder&quot; which the software engineer can follow as he/she gradually learns and becomes more familiar with PSP. RUP has a worker that deals with process improvement issues for the organization. RUP’s focus, regarding software process improvement, is on the organization whereas PSP focuses on the individual.</td>
</tr>
</tbody>
</table>

**RUP definition** A reviewed and approved release of artifacts that constitutes an agreed basis for further evolution or development and that can be changed only through a formal procedure.

**Comments**

**Script**

**PSP definition** Guides the software engineer through a process. The principal elements of a script are its purpose, the entry criteria, the phases to be performed and the exit criteria.

**RUP definition** Computer readable instructions that automate the execution of a test procedure.

**Comments** RUP also has instructions that guide workers following processes, similar to the scripts that PSP applies. But, unlike PSP, these
process guidance instructions have not been labelled.

**Milestone**

**PSP definition** Tasks that will be key indicators of the software engineer’s progress.

**RUP definition** The point at which an iteration formally ends; corresponds to a release point.

**Comments** Unlike RUP, a milestone in PSP does not have to be a point where an iteration ends.

**Checkpoints**

**PSP definition** Checkpoints are based on personal process data. They help the software engineer to follow a procedure.

**RUP definition** A set of conditions that well-formed artifacts of a particular type should exhibit. Checkpoints may also be stated in the form of questions which should be answered in the affirmative.

**Comments** In RUP, checkpoints are derived from common experience, and not derived from workers’ process profiles as is the case with PSP.

**Report**

**PSP definition** Documented results of a process analysis conducted by the software engineer.

**RUP definition** An automatically generated description, describing one or several artifacts. A report is not an artifact in itself. A report is in most cases a transitory product of the development process, and a vehicle to communicate certain aspects of an evolving system; it is a snapshot description of artifacts that are not documents themselves.

**Comments**

**A Comparison of Terms with Different Labels but Same Semantics.** In this section, a number of terms used in PSP and RUP are explained. The terms used in the processes have different labels but same meanings.

**Semantics** Information that states how to perform various activities, and how results of activities performed should be presented. Further, to communicate conventions and idioms used in activities.

**PSP term** Standards
RUP term Guidelines

Comments PSP has standards for design, code and test activities. RUP has guidelines that support more software project activities, such as requirement management, project management and business modelling.

Semantics A unit of work with a definition and a structure.

PSP term Phase

RUP term Activity, Work flow

Comments RUP distinguishes between an activity and a work flow in that an activity is a small unit of work, whereas a work flow is a performed sequence of activities, i.e. a larger unit of work.

Semantics A defined sequence of software engineering process steps required to develop or maintain software.

PSP term Personal Software Process

RUP term Development case

Comments Unlike PSP, which is designed to support an individual developing software, RUP’s development case concerns an organization developing software. The development case is a configuration, or customization, of the Unified Process product and adapted to the needs of a project.

Semantics A distinct sequence of activities with a base-lined plan and valuation criteria.

PSP term Cycle

RUP term Iteration

Comments Cycle and iteration differ in two ways. First, the term cycle, used in PSP, includes only design, code and test activities. An iteration in RUP includes all its work flows, e.g. Business Modelling, Requirements and Deployment. Second, a cycle lasts for a couple of hours usually, an iteration lasts at least for a month.

Semantics Captures the results of a software development activity.

PSP term Log

RUP term Record

Comments
7.5 Incorporating PSP into RUP

7.5.1 Motivation

The expected result of incorporating PSP into RUP is to provide RUP’s workers with ways of improving their software development processes. This section presents arguments why PSP is suitable for this task. The following points motivate the integration:

- RUP only provides limited support to its workers for managing work during an iteration. In RUP, iterations can last for weeks or months. The project manager or other in charge, assigns tasks and responsibilities to project participants. However, it is up to the individual worker to plan his/her work during an iteration. RUP provides little support for this. PSP however, does provide task and schedule planning templates to monitor a project’s progress. This way, the software engineer can realize when he/she is off schedule and inform the project manager about the status and appropriate actions can be taken to correct the situation. If the engineer did not have this insight, problems would probably be corrected later in the iteration, causing more damage.

- Workers in RUP do not perform reviews based on personal defect data. Rather, they have checklists based on the most typical defect types determined by the organization. As a consequence, the software engineers could be looking for defect types they never inject. PSP on the other hand, advocates checklists that are based on personal defect data. The software engineer will be looking for defect types that he/she usually injects, probably resulting in higher review yields as argued by Humphrey [41].

- RUP does not provide templates for writing down ideas for process improvement. PSP has templates which help the software engineer to record ideas and tasks to realize and perform later on in a project.

- RUP does not guarantee to have predefined metrics that software engineers can use to monitor and control their software processes. Before each new project, metrics are defined and explained to all project participants. However, there is no guarantee that the defined metrics offer the kind of information that a software engineer would like to have, in order to monitor the software development process. In these cases, it is up to the software engineer to define and apply appropriate metrics. However, PSP has metrics that provide the software engineer with information how the software development process can be improved. Metrics provided by PSP concern project, product and resource attributes. Thus, PSP offers a set of metrics that provides adequate
support to monitor and control the software development process at an individual level.

- As a last argument but not the least, the spirit and soul of PSP should be mentioned. PSP stresses the fact that personal commitment and excellence are factors affecting process improvement results. An employee with this commitment probably outperforms, in the long run, an employee with little interest in improvement issues.

7.5.2 Procedure

The procedure of incorporating PSP into RUP consists of two parts. First, the PSP material, i.e. phases, logs, templates and metrics is allocated to appropriate workers in RUP. Further, guidelines for modifying, introducing and maintaining PSP material is provided. However, how these guidelines are followed may differ from project to project due to situational factors in the projects and is beyond our sphere of control.

PSP has not changed its process contents since it was introduced in 1995. The RUP version used in this work is the latest release at the time when this work is conducted, i.e. RUP 2000. However, earlier releases of RUP do not seem to differ too much regarding the contents of the process. This means that organizations using older versions of RUP could likely apply the ideas presented in this work.

The procedure of incorporating PSP into RUP is divided into three steps, presented below:

1. **Select workers in RUP.** In RUP, about 30 workers may be involved in a software project. Each worker represents a role that is responsible for producing artifacts and carry out activities. Since PSP is an SPI framework at an individual level the strategy is to identify workers in RUP that benefit from using PSP. This way, the disseminated PSP material would still be applied at an individual level, which it was designed for. The author of this thesis analyzed and categorized the contents in PSP against every worker in RUP, to sort out the workers of interest. This resulted in five workers which received PSP material.

2. **Allocate PSP Elements.** To ensure that each selected worker received PSP material that fitted the worker’s work profile it was necessary to perform a detailed review of each worker. Further, it is also important to ensure that each worker receives PSP material that are not already covered by original artifacts from RUP. Both RUP and PSP have artifacts or elements, i.e. output of an activity like Design Review Checklist or High-level Design, and it may be that the worker in question has an artifact similar to one provided by PSP, and thus is not in need of the element from PSP in question. Hence, the detailed
comparison of the selected workers’ artifacts and PSP’s elements was motivated.

3. **Provide Guidelines for Introducing PSP.** RUP and PSP differ in several aspects regarding software development, e.g. artifacts, activities, and number of people involved. For PSP elements to support the RUP workers, they must suit the RUP context. Further, every RUP project is unique and has its own needs. Therefore, guidelines of how to adapt, introduce and maintain PSP in the RUP context are presented. No predefined material was presented (e.g. templates and metrics) since RUP projects can differ in many ways (e.g. use of life cycle model and definitions of artifacts). As a consequence, every RUP project should adapt the PSP process to its own needs and apply it accordingly. The actual modification of PSP will be performed by individuals participating in RUP projects.

The first six levels in PSP apply the waterfall life cycle, it is only the last level, PSP3, that applies an iterative life cycle. However, since RUP applies an iterative life cycle all levels in PSP need to be iterative and were adjusted accordingly. As a consequence, PSP3 was left out from the incorporation. Further, PSP2.1’s main contribution is the use of design templates. However, since only the workers Designer and Test Designer apply those design templates, they were placed in PSP2. This way, every worker is provided with the same number of maturity levels. Thus, PSP2.1 was also left out when incorporating PSP into RUP.

Below, the three main steps of incorporating PSP into RUP are presented.

1. **Select workers in RUP.** To select workers in RUP that could receive PSP material, every worker’s job description was reviewed and categorized according to the PSP material. The workers who received PSP material performed similar software developing activities as included in PSP, i.e. design, code and test. Some workers performed similar activities among themselves. In these cases, the worker whose work assignment best fitted the PSP material was chosen. The other workers can, to some extent, apply PSP material that the chosen worker received. However, since they are not the best candidates to receive PSP material, they were not selected. Figure 7.1 displays percentages of workers in RUP that do not apply, apply and apply to some extent PSP material.

The selected workers that received PSP material are presented in Figure 7.2.
Figure 7.1: Percentages of RUP workers involved in PSP activities.

Figure 7.2: PSP disseminated to RUP workers.

2. **Allocate PSP Elements.** The allocation of phases, logs, templates and metrics to RUP workers followed a similar procedure like the one applied when selecting appropriate workers. The selected workers’ job tasks were studied and categorized with regard to the PSP material. In the case when several workers can apply the same PSP element, they do that. The intention for incorporating PSP into RUP is to provide the RUP workers with ways of improving their software development processes. This means trying to provide each worker with maximum support. Further, RUP may have artifacts identical to PSP elements, as is the case with RUP’s Programming Guidelines and PSP’s Coding Standard. In these cases, the artifacts from RUP are used since the workers are familiar to them. As a consequence, the mapping between PSP elements to RUP workers is not

\[ 1 \Rightarrow 1, \quad \text{but} \quad 1 \Rightarrow N, \quad N \in 0, 1, 2, \ldots, N \]
The parts to map were:

**Phases.** Planning and postmortem phases are used by all workers. Therefore, these phases are filled with black, indicating that they are applied by all workers. Figure 7.3 shows the mapping between PSP development phases and RUP workers. A white circle refers to a phase that is not applied by all workers in the figure. A black circle refers to a phase that is applied by all workers in the figure. A rectangle refers to a worker. An arrow shows a phase being applied by a worker.

![Figure 7.3: Mapping PSP phases to RUP workers.](image)

**Logs and templates.** Some logs and templates in PSP support specific phases in software engineering, other support process improvement in general. As a consequence, some logs and templates are applied by all workers, while some are applied by only one or a few workers. Figure 7.4 shows the allocation of logs and templates to the workers. In the figure, a rectangle refers to a worker/workers that apply the logs and/or templates written inside the rectangle. "All workers" implies that the logs and templates inside its rectangle can be applied by all of the chosen RUP workers. The number to the left of each log or template refers to the PSP level when the log or template is introduced.

**Metrics.** In software development, there are three major metric categories: products, processes and resources as argued by Fenton
Figure 7.4: PSP applied by RUP workers.

and Pfleeger [29]. Product metrics refer to the volume of the product produced. Process metrics provide information about the behavior of the process and resource metrics track the time spent in a software project. PSP has metric types that fall into all three categories.

As with the logs and templates, some metrics refer to specific software engineering tasks while others can be applied generally for improving the software process. Figure 7.5 presents how PSP metrics are disseminated to selected workers. In the figure, the tables should be viewed as an inheritance tree where the root is at the top. Thus, the worker Tester can apply all the metrics that All workers can use, plus the metric Test Defects/KLOC and so on. The column headers, PSP0 through PSP2 indicate the PSP levels where the metrics in question are introduced.

To integrate implemented components may not necessary mean writing a lot of code. Therefore, metrics like Reused and LOC/Hour will likely have low values for the worker Integrator. However, these values only concern the Integrator which uses them to analyze his/her process. They should not be compared to any other worker.
3. **Provide Guidelines for Introducing PSP.** The intention of this work is not to create modified scripts, logs and templates ready to be applied when using PSP in RUP. Instead, guidelines are presented to
organizations that use RUP of how to adapt, introduce and maintain PSP in RUP projects. That way, organizations that use RUP and are interested in PSP can incorporate PSP into RUP.

There are two different levels at which a software engineering process can be modified, according to RUP [57]. The levels for modification are organization wide and project specific. The organization wide process is a common process used throughout the organization that defines core technologies, reuse practices and so on. The project specific process refines the organization wide process for a given project. Here, parameters such as size of a project and initial cycles affect the definition of a process. A project specific process in RUP is called a development case. It describes the development process that an organization has chosen to follow in its project. To help customizing PSP to a RUP project, the PSP material may be adapted in parallel with the definition of the project’s development case.

Oskarsson and Glass [72], discuss the notion of a quality system for software. A quality system is defined as "organizational structure, responsibility, routines, processes and resources for leading and controlling the ongoing work with respect on quality". Following this definition, both PSP and RUP can be seen as quality systems. Further, some factors that facilitate the incorporation and maintenance of a quality system into an organization are presented. Since PSP and RUP can be classified as quality systems, these factors are valid and may prove useful to consider when incorporating PSP into RUP. The factors are according to Oskarsson and Glass [72]:

**Involving the right people.**
When introducing software development routines, it is important that right people are involved in the work. Appropriate persons are usually competent individuals in the organization that work in the area of interest.

**Documenting new routines.**
All routines, especially new ones, should be documented. In larger organizations there is important to clarify who will do what and when. Putting the routines on print makes this easier. Further, software engineers can use written routines as a reference manual when learning new routines.

**Conducting quality revisions.**
There should exist internal quality revisions that check if co-workers apply the defined quality system appropriately. The purpose of an internal quality revision is to find areas in processes that can be improved. For instance, if employees do not use a certain routine it may be a sign that the routine should be
7.6. CONCLUDING REMARKS

changed.

Feedback on process usage.
Suggestions and complaints from people applying a quality system should be considered. Everyone involved in a project should understand that feedback from applying a quality system is welcome. Preferably, some sort of communication channel should be established that facilitates the communication between management and workers.

Postmortem-analysis.
After some kind of milestone has been reached, an analysis of the conducted work should be done and appropriate improvement actions taken.

The factors mentioned above help organizations to adapt, introduce and maintain PSP in RUP projects. However, the factors are general in nature and apply on any kind of quality system. To further support the incorporation of PSP into RUP, more information about how RUP defines development cases is needed. RUP has numbered a set of factors that should be considered when defining a development case in RUP [57], presented in Table 7.7. Comments follow each factor about how it may affect adaptation of PSP.

7.5.3 Improving RUP with PSP

RUP provides little support to its workers in improving the way they work. The worker Process Engineer tries to improve the working process within the organization. As a consequence, the process engineer may order another worker to change his/her working behavior which may improve the worker’s development process. However, there is a difference in taking orders or performing personal analyzes of how one’s process should be changed. This is where PSP comes in. It helps individuals realizing how they should change their software development processes in order to improve them. Table 7.8 presents areas of the software development process in RUP where PSP can help to improve.

7.6 Concluding Remarks

This work presents the incorporation of PSP into RUP. An allocation of PSP material to selected RUP workers is presented. Further, guidelines how to adapt, introduce and maintain PSP in RUP projects are presented. This helps organizations that use RUP to adapt PSP to their own needs and apply it accordingly.
Table 7.7: Guidelines for introducing PSP into RUP.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze the project and the organization.</td>
<td>Contains information about the project and the organization. Here, issues like communication in a project affects layout of PSP forms. Further, choice of size metric for the software development effort is important, PSP uses LOC. The use of another size metric in a RUP project means changes in PSP.</td>
</tr>
<tr>
<td>Define scope.</td>
<td>Determines among other things which work flows that will be applied in a project. For instance, if work flows involving design, code and test are not included, then the PSP contribution will be less.</td>
</tr>
<tr>
<td>Decide how to perform each core work flow.</td>
<td>Activities in PSP that are included in a RUP project should be modified to suit the RUP context before they are applied.</td>
</tr>
<tr>
<td>Tailor artifacts per work flow.</td>
<td>Based on the contents of a work flow and how activities are performed, artifacts are tailored to suit these demands. This also applies to PSP elements.</td>
</tr>
<tr>
<td>Choose life cycle model.</td>
<td>The choice of life cycle model affects which phases in PSP that are included in a RUP project.</td>
</tr>
<tr>
<td>Describe sample iteration plans.</td>
<td>A sample iteration plan describes how the project should be run in different iterations and phases. PSP activities should also be included in the plan.</td>
</tr>
<tr>
<td>Document the development case.</td>
<td>PSP routines should be included in the documentation of the development case. This enables project participants to refer to PSP when applying it in a RUP project.</td>
</tr>
<tr>
<td>Maintain the development case.</td>
<td>The software development process should be evaluated after each iteration. Apart from evaluating use of PSP, the integration of PSP and RUP should be reviewed. This way, PSP’s support to RUP is under constant improvement.</td>
</tr>
</tbody>
</table>

An aspect in this work that may receive critique is the rather loose approach for how selection of RUP workers and allocation of PSP material
### Table 7.8: Areas in RUP where PSP may provide support.

<table>
<thead>
<tr>
<th>Area in the software development process</th>
<th>Motivation for improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design and code reviews.</strong></td>
<td>Reviews conducted in PSP are based on personal defect data. This means that the engineer looks for defects that he/she injects, not what others inject. As a consequence, a higher percentage of injected defects are probably found at the reviews as argued by Humphrey [41]. This saves time and money as less defects are found later in the development life cycle.</td>
</tr>
<tr>
<td><strong>Monitor and control of the software development process.</strong></td>
<td>PSP provides support for the software engineer to realize opportunities for improvement of the software development process. By collecting and analyzing process data, can the software engineer realize which areas in the software development process that need improvement. The templates store process data, which provides information to improvement issues.</td>
</tr>
<tr>
<td><strong>Estimation.</strong></td>
<td>Estimates done in PSP are based on personal data. This means that an estimation done by a software engineer is based on the work effort of the engineer. As a consequence, if the prerequisites are the same, e.g. same programming language and problem domain, the estimations will be better than estimations based on data collected throughout the organization as stated by Humphrey [41].</td>
</tr>
<tr>
<td><strong>Iterative development.</strong></td>
<td>PSP offers support to plan and monitor work in short iterations. The iterations are shorter then the iterations conducted in RUP. This means a way for the software engineer to constantly know status of his/her process. Any deviations from the original Iteration Plan can then be detected and appropriate actions taken.</td>
</tr>
</tbody>
</table>

were conducted. It has been motivated why the steps in the procedure are necessary but not how the steps are performed. The way to perform
these steps meant using our subjective judgement due to limitation of time and resources. It would have been theoretically possible to have performed some kind of formal analysis where other people which understand RUP and PSP, could have disseminated PSP material to RUP workers and analyzed their results and compared it to our result. However, since people with appropriate backgrounds were difficult to find was this approach not applied.

The question "Which workers will lead the task of incorporating PSP into RUP?" has been left unanswered in this work. The reason for this is simply that the author of this thesis believes no definite answer exists. Our opinion is that persons responsible for incorporating PSP into RUP should be motivated people which have an interest in the subject. An important factor when introducing new routines is to involve people that are going to use the routines, as stated by Oskarsson and Glass [72]. Just ordering a person to be responsible for the incorporation is probably not the right way to go. Management of a RUP project should learn the project participants’ attitude towards the incorporation and allocate responsibilities thereafter.

This study shows that a process may be adapted by receiving contents from another process. In this case, a number of workers in RUP received support from PSP. This solution has the benefit that the introduced support originates from a defined, formal process where the contents have been validated and thus constitute a sound basis to start improving the software development process. It is important to remember that a process should be adapted to the specific situation or environment where it will be used and not the other way around. This notion is for instance embedded in PSP where the name itself, Personal Software Process, stresses the need to adapt a process to make it personal so that it supports the needs of the actual person using the process.

There exist a number of workers in RUP not involved in software development areas which PSP addresses. As Figure 7.1 shows, about three quarters of the workers in RUP are not involved in PSP-related activities. A PSP specially defined for each of them, would provide them as well with frameworks for improving their software engineering skills. RUP has a number of workers, not involved in PSP-related software areas, that are active early in a project. By improving their software engineering skills, they would produce better work results early in the software development life cycle. This would create a more solid foundation for further work in a project. This in turn would save time and money considering the amount of time needed to locate and fix defects late in the software development life cycle, compared to early as reported by Humphrey [41]. This issue, of developing an SPI framework at an individual level in areas where PSP-related activities are not involved is addressed in Study 2.
Part III

Support for Agile Methods
Chapter 8

INTRODUCTION TO STUDY 4

8.1 Overview

This chapter includes a summary of the purpose of Study 4, which is a case study consisting of three separate studies where a team is introduced to a process based on XP. Further, an overview of the team’s research environment is provided. Then, the introduced process is compared to how the team used to develop software prior to the introduction of the agile method. A general comment when reading Chapters 9 to 12 is that it helps if the reader is familiar with XP and its terminology. More information about XP is provided in [10].

8.2 Background

A team was introduced gradually to an agile method based on XP. The method was adapted to fit the team’s development needs. The purpose of the case study is to investigate the results of introducing XP to a large software development organization. There are reports on similar efforts provided by Murro et al. [67] and Grenning [38]. However, it is our opinion that these reports do not adequately describe what has been changed in the software development process as a result of introducing XP, and how the development environment was before the introduction of XP. This information is important in order to understand the underlying reasons for any improvements. Further, the results from many reports such as Murro et al. [67] and Grenning [38] are based on subjective experience, which is important but only portraits one side of the results. This chapter provides a detailed description how the team worked prior to the introduction of the agile method, what was introduced and how. Further, Study 4.1 to Study 4.3 present results based on statistical analysis as well as subjective expe-
Experience in form of interviews and observations. Thus, the intention with the studies is to provide a detailed picture of the results of introducing an agile method based on XP to a large software development organization.

As these studies were conducted in parallel they shared the same research context. By research context, it is meant how the team used to develop software before the introduction of the agile method (e.g. programming languages, test procedures and so forth) and how the team changed its process as a result of using the agile method. Further, it is important to describe what parts of the agile method that was introduced and how these parts were adapted in order to understand the cause for any improvement or deterioration in the team’s performance. Thus, the motivation for describing the context in detail is based on three factors:

1. To realize how the team developed software prior to the introduction of the agile method.
2. To realize how the agile method was introduced and adapted.
3. To realize how the team changed its way to develop software, as a result of the introduction of the agile method.

An improved knowledge of these issues will help evaluate the results from Study 4. To investigate the validity of the results from the case study, the conclusions from the case study were addressed in a survey where a number of organizations shared their experiences of introducing agile methods based on XP. The results of this survey is reported in Study 5.

8.3 Study Context

In this section, the participants of the study and their development environment are described. Prior to conducting the case study the author of this thesis worked at the company where the case study was conducted. Thus, a motivated question to ask is if and how the author’s role at the company affected how the case study was conducted. It is our opinion that the work conducted at the company prior the start of the case study did not affect how the case study was conducted. The reason for this is that the author did only work with a limited number of people. Thus, the author did not have any sort of reputation in the company which may have affected how the case study was conducted.

8.3.1 The Company

The study was conducted at a Swedish software development company. About 1500 people are employed by the company. Out of these 1500 approximately 250 work as software developers. Company X, sometimes referred to
as the case company, develops and maintains more than 30 software systems with several code tracks. In addition to this, a number of software releases are delivered to customers each year. Apart from developing new functionality the company also corrects, adapts and modifies software systems. Based on these characteristics (i.e. number of maintained systems and development activities), Company X is classified to have both a complex and an evolutionary and maintenance software development environment. Complex in the sense that over 30 systems are maintained and integrated in different releases each year. Further, these systems are maintained by Company X which means adding and removing functionality to the systems. Thus, the systems evolve in both purpose and functionality as a result of the maintenance efforts.

8.3.2 Study Participants

The participants were all vaguely familiar with agile methods but did not have any previous experience of any formal software development process. At the start of the study, the team consisted of six developers and one team leader. The team leader had the XP roles coach and tracker. The developers’ age ranged from 24 to 42. Everyone in the team was educated at postgraduate level with degrees of Master’s of Science or similar. Number of years as a programmer ranged from 5 to 18.

8.3.3 Development Environment

The team developed new functionality and maintained a large software system. The system could be described as a transport layer for transactions. The system was used to connect other systems, in order for them to communicate. Thus, the system that the team maintained had an important objective as it provided a communication infrastructure for the other systems. The system was integrated in at least four major releases each year, where the company had recognized needs from its customers and updated the systems accordingly. The chief architect, who had developed the kernel of the system, worked in another country and regularly communicated with the team how to maintain and update the system. Important aspects for the team to address were reliability and fault management. Due to the system’s nature of supervising transactions, part of the testing concerned communication over the internet. For this, the team had a test environment where network problems such as hardware failure, power cut failures and so forth could be simulated. Further, the system’s client applications supported a number of operating systems, thus requiring a certain level of competence from the software engineers. The software engineers used the programming languages C and C++. They had Visual C/Studio as their development environment.
The team divided its time between updating the system with customer requirements, developing new functionality or improving the overall reliability of the system. The team received its implementation tasks in form of so called functional specifications which consisted of detailed information of what the team should implement. These functional specifications were delivered by the project manager in charge of the actual project that the tasks addressed. The team leader planned the team’s work with project managers which the team worked for. The software developers’ work tasks consisted of design, code and test activities. However, major test activities such as system tests were performed by the company’s test department. The team documented inserted changes in the code. The documentation activity was regarded as necessary since it helped understand the purpose of the code and how the different code modules interacted.

Before the agile method was introduced, the team was dispersed in four rooms. Thus, the communication was not so prominent as it was when the agile method was introduced. The team had daily meetings were they discussed work tasks for the day. They performed test activities, but not so often as was the case when the process was introduced. They followed implicitly stated coding standards that existed in different systems. That is, they followed the structure of the code they made changes in. They did not have a defined coding standard of their own. Most of the time, they solved their tasks individually, but when needing help to solve an issue they talked to other team members for help. During the study, there was minimal development of new functionality. Most issues concerned support, usually meaning vague problem descriptions making estimation of tasks more difficult. In the support phase, many tasks emerged during iterations. These incoming tasks often had high priority and needed immediate attention.

8.4 Introducing the Agile Method

Company X where the study took place had increased its focus on delivering high quality software. The team did not apply a formal software development process, before the introduction of the agile method. The team applied an informal approach to software development. Although informal it contained discipline as the team followed certain procedures such as conducting design meetings with customer representatives and documenting inserted changes in the code. However, the team was of the opinion that by following a formal process its software development performance could be increased. Thus, a number of software development processes were considered which would help the company improve its software development process. A requirement was that the process should be easy to introduce and use, and have an ability to successfully address sudden changes in software development. Hence, different agile methods were considered but XP was chosen
since it was the most prominent agile method, and it provided concrete
guidance through the use of its practices.

As stated above, the introduced process was based on XP but not iden-
tical to it. This depends on several factors. First, some practices were not
applied since they did not comply to the company’s business culture or they
were not appreciated enough by the team or higher management in the com-
pany. Second, some practices were adapted to fit the team’s development
environment. Thus, the main part of the practices were not applied as XP
advocates or not introduced at all. A large part of XP did not affect how the
team developed software. Thus, efforts were spent on introducing the parts
of XP that generated changes in the team’s software development process.
One practice at a time was introduced because the team was busy and not
ready to start use all of the agile method at the same time, and the team
thought it was a good way to learn the process.

The length of an iteration was two weeks, which is by many applying
XP the recommended iteration length. Further, the team thought two weeks
was appropriate referring to its development situation. At the start of an
iteration, a new practice was introduced. The practice Planning game is
important to introduce early as it concerns planning of work in iterations.
As a consequence, Planning game was introduced first. Then, more prac-
tices were introduced and adapted to fit the team’s software development
environment.

There exist different views in the agile community how to introduce XP,
either one practice at a time, a couple at a time or all practices at once.
Some argue that it is important that all of the practices are introduced
simultaneously in order to receive maximum benefit of the method, while
other argue that it is better to introduce the practices gradually. However,
there does not exist any scientific evidence regarding which approach is best.
In this study, one practice at a time was introduced as the team was busy
with regular work and the team would have problems to learn, adapt and use
all of the practices at once. Further, considering the team’s busy schedule it
was seen as more pedagogically to introduce one practice at a time. Thus,
there was a joint decision to introduce the process gradually. The procedure
for introducing the process was to inform the team of a practice before it was
introduced to the team. This way, the team could discuss internally whether
it perceived if the actual practice would help the team improve its software
development process and thus decide whether to introduce the practice or
not. Further, at these discussions the team also decided if and how the
introduced practice would be adapted. Before introducing a new practice
the team was informed what the purpose of the practice was and discussed
if the practice should be adapted in some way to fit the team’s development
environment. Further, adaptations of the practices evolved over the study.
That is, the implementations of practices were not freezed at any time, on
the contrary the team was allowed to adapt them as often as they wanted
The practice *Metaphor* was not introduced since the architecture in the system was thoroughly defined and used throughout Company X’s development organization. Hence, an introduction of a metaphor would cause more harm than good. The practice *40 hour-week* was not introduced since the team thought the practice naive and not applicable to its development environment. That is, they could not guarantee to follow the practice if a customer needed attention. Further, it was not uncommon for the team members to have at least a couple of hours overtime each iteration. The practice *Refactoring* was not introduced. The team did not have sufficient support for regression testing which made them reluctant to make changes to the legacy code in fear of inserting new defects. Further, management did not support refactoring activities due to an earlier technical mishap associated with a refactoring activity.

The text is structured as follows. First, the team’s original process is described in XP terms. The purpose of this is to enable a comparison with the introduced process, which is based on XP. However, Section 8.3.3 contains information regarding how the team used to develop software prior to the introduction of the agile method which are difficult to cover in XP terms. Thus, it should be mentioned that the comparison does not cover all aspects of how the team used to develop software. Then, changes in the team’s way to develop software are identified as a result of introducing the agile method. Finally, the section contains a comparison between the introduced process and XP to clarify how the processes differ in content.

### 8.4.1 Original Process

This section presents how the team developed software before the agile method was introduced. The team’s original process is described in terms of XP, as the original process later is compared with the introduced process to identify process changes. XP consists of twelve practices that offer a structured approach to software development. Below, the original process is described in these practices.

**Planning game.** The team planned what functions should be included in coming software releases. This was a rough plan often covering several months. The team had daily morning meetings where they discussed what each person did, and coming work of the day. Here, tasks were chosen depending on what the team leader prioritized. The team members did not plan their work in detail, instead they solved issues as they came.

**Pair programming.** The team members solved problems individually but asked other team members when there was a need to. The team mem-
bers sat close to each other, in adjacent rooms which facilitated communication.

**Collective code ownership.** All of the team members were encouraged to make changes in the code. However, it was common that the person with the highest knowledge of a part of the system where changes were made, inspected the new code for correctness.

**Small releases.** The team produced new releases based on its own initiative. The team usually made 1 to 2 releases per week.

**Coding standard.** The team worked with several systems, where each system had its own informal coding standard. These informal standards were not written down on paper but existed informally.

**Simple design.** The team could not do the simplest solution possible, since they had to plan for how new code would interact with coming functionality. Future requirements were predictable, which makes simple design difficult to apply. However they tried whenever possible, to reduce the complexity of a design.

**Metaphor.** The team worked in a complex software development environment, maintaining a large number of systems. The company had an established architecture for the systems which was described in various design documents.

**40 hour-week.** The team often worked overtime, depending on the work load. The motto was to keep the customer happy, which could mean work even at weekends to solve customer related problems.

**Refactoring.** The team did not perform any refactoring activities. This decision was based on two aspects. First, the team managed a lot of code and changes in the code may cause new defects. Further, the team did not have sufficient support for regression testing. This made the team reluctant to make changes to legacy code in fear of inserting defects. Second, management did not support refactoring activities. As a consequence, the team did not have formal approval of conducting refactoring.

**Continuous testing.** The team performed unit tests, system tests and acceptance tests. Acceptance tests were conducted at special occasions when real customers were present. Often, team members helped customers install systems and perform tests to ensure customer satisfaction. The team did not have a unit test framework. Thus, no suit of unit test cases were run.
**On-site customer.** The team did not have a real customer participating in the team. They discussed development issues with customer representatives working at the case company. The team had several customers at once. Contacts with the customer representatives occurred at planned meetings. The team received requirement specifications from the customer representatives. Thus, they did not discuss in detail with a real customer how functions should be developed.

**Continuous integration.** The team did not integrate the system as soon as changes were made, since this was a complicated and time-consuming activity. Instead, the team compiled new code locally on its computers and typically once a week was the system integrated.

### 8.4.2 Introduced Changes

The agile method was based on XP. Practices in XP were introduced to the team who adapted them to fit its development environment. Table 8.1 presents practices that changed the team’s software development process. The comparison concerns the team’s process before versus after introducing the agile method.

### 8.4.3 Comparing the Introduced Process to XP

Table 8.2 presents differences regarding the agile method versus XP. The table contains the practices that were implemented differently than XP advocates.

As mentioned in Section 8.4, nine out of XP’s twelve practices were introduced and out of these nine practices four were regarded to significantly change the team’s software development process. Thus, the conclusions from Study 4.1 to Study 4.3 are based on the impact from these four practices. It should be mentioned that the team’s choice not to apply certain practices may have a negative effect on the quality of the developed software, in a long term basis. For example, the practice *Refactoring* is probably useful to maintain a well-organized structure of the code. Further, some of the introduced practices did not significantly change how the team developed software. A more strict application of how the practices are supposed to be used may have generated more positive effects. For example, the practice *Pair programming* was implemented in such a way that two people met before and after a task was implemented. If the team had decided to always develop software in pairs, then maybe the team’s overall competence would have increased even more as a result of developing software in pairs. However, the standpoint in the research was to introduce XP to a team in the industry and adapt it with respect to the team’s needs and opinions. The term ”based on XP” is used throughout the thesis as the purpose was to introduce, adapt and use XP in a real software development environment,
8.4. **INTRODUCING THE AGILE METHOD**

Table 8.1: Process changes due to introduction of the agile method.

<table>
<thead>
<tr>
<th>Before Agile Method</th>
<th>After Agile Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planning game</strong> did not exist. Future work was not planned in detail. Daily tasks were given by the team leader to the software developers. Development tasks were time estimated by people outside the team. The team lacked an overview of past, current and future work.</td>
<td>Planning game existed. Future work was planned in detail, in two-weeks periods. Development tasks were chosen by the software developers. Development tasks were time estimated by the team. The team had an overview of past, current and future work.</td>
</tr>
<tr>
<td><strong>Pair programming</strong> was not applied. The software engineers solved software issues individually. However, contacts occurred with other team members when help was needed to solve a development task.</td>
<td>Pair programming was partly applied. The software engineers solved development tasks individually. However, pairs met at least twice per task. First, to discuss the problem of a task. Then, to present the solution.</td>
</tr>
<tr>
<td><strong>Continuous testing</strong> occurred frequently although in an unstructured manner.</td>
<td>Continuous testing occurred frequently in a structured manner, as unit test cases were continuously added and run when new code was added.</td>
</tr>
<tr>
<td><strong>On-site customer</strong> did not exist. The team planned its work by itself.</td>
<td>On-site customer existed, although the practice was adapted. The on-site customer planned the team’s work in two-weeks periods.</td>
</tr>
</tbody>
</table>

and in such an environment it is not always possible to apply XP exactly as intended.

Previous to this study an attempt to introduce XP to another team at the case company was made. The purpose of the study was the same as with this study. But, due to a reorganization the study was ended. However, this first attempt provided experience regarding conducting process change at the case company. Further, the author of this thesis had worked for one year at the case company and participated in internal courses of the case company’ systems. Thus, the software development situation was familiar. Before introducing XP, the study participants were interviewed of their current way of developing software. Thus their software development process was known before the agile method was introduced. These aspects improved the ability to analyze the research results, as they could be analyzed with regards to what and how the team’s software development process was changed.
Table 8.2: Differences regarding the agile method versus XP.

<table>
<thead>
<tr>
<th>Introduced Agile Method</th>
<th>Standard XP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planning game</strong> does not include user stories. No customer is present during planning game. No release planning occurs.</td>
<td>Planning game includes user stories. Customer is present during planning game. Release planning occurs.</td>
</tr>
<tr>
<td><strong>Pair programming</strong> is adapted. Software tasks are solved individually, but pairs meet regularly to discuss results.</td>
<td>Software is developed in pairs. Two persons shift places periodically, one writes and the other thinks strategically.</td>
</tr>
<tr>
<td><strong>Coding standards</strong> are used for different systems.</td>
<td>One coding standard is used for all software development.</td>
</tr>
<tr>
<td><strong>Simple designs</strong> may include needs for future integration of new functionality.</td>
<td>A simple design include what is needed for the moment.</td>
</tr>
<tr>
<td><strong>Continuous testing</strong> covers only part of a system.</td>
<td>Continuous testing covers a whole system.</td>
</tr>
<tr>
<td><strong>On-site customer</strong> is an internal customer representative, and is not part of the team and not available all the time.</td>
<td>On-site customer is a real customer and is part of the team and available all the time.</td>
</tr>
<tr>
<td><strong>Continuous integration</strong> of a system occurs at least once a week.</td>
<td>Continuous integration of a system occurs at least once a day.</td>
</tr>
</tbody>
</table>
Chapter 9

STUDY 4.1: QUANTITATIVE MEASURES ON AGILE DEVELOPMENT

9.1 Background and Problem

According to Beck [10] the ideal environment for introducing XP is in a small to medium sized software development project. The team applying XP should consist of about 4-12 people developing new functionality with one customer. Thus, XP is applicable in small and simple software development environments. Further, since a system should be integrated, built and tested several times a day with XP it should not take long to compile and link the system. In addition to this, it is important to have the right physical environment. However, this ideal development environment does not reflect the actual development environments in most software companies. On the contrary, according to our experience many software development organizations conduct projects with complex environments with, for example, several customers and a number of systems to maintain. Often, these environments include development, evolution, maintenance and support activities. Further, organizations may have problems to fulfil the right physical environment that XP advocates due to different reasons. These aspects make it hard to introduce and apply XP as described by Beck [10]. Thus, the contents of XP was adapted to fit the development situation in the case company (described in Chapter 8) where it was introduced to.

The XP approach was introduced to a complex software development environment, which meant that not all of the practices were introduced. Further, some of the practices that were introduced were adapted to fit the company’s software development environment. In this kind of environ-
ment it is necessary to adapt the XP practices differently than to the ideal environment described by Beck [10]. Thus, the study investigated results when introducing an agile process, based on XP, in an environment that is representative to many large software development organizations.

A number of suggestions are reported by Beck [10] on how to improve software development results by applying the XP approach. However, unfortunately little or no data is provided to support these recommendations. Hence, more research is needed regarding the issue. Thus, it is interesting to investigate how XP is implemented and what the results are in these real world environments. This study reports quantitative results when introducing an agile process based on XP to a complex software development environment.

Section 9.2 contains related work and Section 9.3 presents the research methodology. In Section 9.4, the research results are presented. Section 9.5 includes an interpretation of the research results. Section 9.6 summarizes the main conclusions from the study and Section 9.7 discusses the research results.

9.2 Related Works

Maurer [62] reports on a case study which compares the productivity resulting from applying practices from XP with a more conventional object-oriented software development approach. Four productivity metrics are used, and the results indicate productivity gains when applying XP. Productivity is defined as size divided by effort. For size, four factors are applied: *Number of new lines of code in a given release*, *Number of new classes in a given release*, *Number of features added and bugs fixed in a given release*. Effort is defined as the *Number of hours billed per month*, as the company participating in the study believed that best reflected the actual amount of work done. The results were positive, three of the metrics showed an increase in productivity. The metric *Number of new lines of code in a given release* increased with 65% after applying parts of XP. The metric *Number of new methods in a given release* increased with 302%, and the metric *Number of new classes in a given release* increased with 283%. However, the metric *Number of features added and bugs fixed in a given release* decreased with 5%. The authors argue that this number is misleading since a change in software made it more difficult for the software engineers to handle the administrative work when closing defects. Overall, the study indicates a productivity gain for a small company when applying XP.

Putman [78] reports improvements from applying XP in a company called Workshare. Data over a two year-period was collected by the company, from March 2000 to February 2002. XP was introduced after the
first year. Hence, it was possible to compare results from applying XP with results from applying the previous ad-hoc software development process. Workshare’s implementation of XP meant that eight out of twelve practices were applied. The practices not applied were Coding standard, 40-hour week, Simple Design and Metaphor. Prior to the introduction of XP, Workshare spent 70% of their development effort removing defects, and 30% on new development. After introducing XP, they reduced their defect rate about 80% thus only spending 14% of the development effort removing defects and 86% on new development. This indicates an increase of around 300% in productivity. Further, Workshare applied a metric called Support Calls - Average Time. The metric was defined as the number of days a call had been opened. Before introducing XP, measurement results were averaging 70 days. After XP was introduced, measurements results were averaging 20 days.

9.3 Research Methodology

The purpose of this study is to investigate the results of introducing an agile method to a complex software development environment. Complex means an environment that significantly differs from the ideal XP environment described by Beck [10]. The study was conducted at a Swedish software development company, described in Chapter 8. In general there exist few quantitative studies on introduction of XP, which in itself motivates a quantitative study on agile development based on XP. But, an even stronger motivation for us was to investigate what software process improvement results could be seen when introducing a process based on XP to a complex software development environment, since these environments exist in many software companies. In the study, one of the case company’s software development teams’ performance was measured during the length of the study. Statistical analyzes were performed on the results gained, and conclusions drawn based on the results of the analyzes. The research question is formulated as:

Does the introduction of an agile process to an evolutionary and maintenance software development environment improve software development performance?

As discussed below in Section 9.4 results were compared from using the agile method with historical data when the agile method was not applied, in order to realize the effect the agile process had on the team’s software development performance. By using historical data, a baseline could be established concerning the team’s performance without using the agile method. Preferably, the team would be more productive and efficient using the process since this would indicate that it helped the team improve its way of
work. Some metrics in the study are strictly related to XP. The metrics are Project Velocity, Load Factor and Planning Game Tasks, Percentage, presented in Section 9.3.4. For these metrics it was not possible to produce historical data where the agile process had not been used. In these cases, data was analyzed by applying linear regression analysis.

The case environment did not fulfill the type of XP environment advocated by Beck [10]. Thus, it was not obvious that the software engineers would perform better using the agile method. The null hypothesis was that there would be no difference in results when using the agile process compared to when not using it.

9.3.1 Design

A fixed research design approach described by Robson [84] was used for the research. Fixed means a design where the researcher in advance has decided research decisions such as what to measure, how and when. This is important in quantitative studies since in order to detect trends in data, the same metrics should be applied.

The research method used was a quantitative, explorative case study as defined by Robson [84]. The study was explorative in character since software process improvement results of introducing an agile process to a complex software development environment was investigated, which the author of this thesis did not have any prior experience of. A case study was appropriate since the study was an empirical investigation of a particular contemporary phenomenon within its real life context. That is, applying an agile process at the case company under a limited amount of time.

9.3.2 Data management

This section presents how the data was managed in this study.

Data Collection. Five metrics were collected. After a completed iteration, the team manager reported data to us. An iteration is a time period used in XP, usually between 1 to 4 weeks, where the XP team is assigned work tasks to be delivered during the iteration. An iteration can be viewed as a mutual agreement between customer and team what functionality is to be delivered in a time period. The data was reviewed and, when necessary, the team manager contacted to confirm its validity. After the data was reviewed, it was stored electronically for further analysis.

Data Analysis. Statistical tests were performed on the collected data. The measurements show the team's performance when using the agile process. The measurements both compare historical data without using the agile method to collected data using the agile method, and they
present trends in data for the metrics strictly related to XP where no historical data without using the agile process can be collected. These trends concern data for all the completed iterations in the study.

When comparing results without using the agile method versus using it, the tests used were the Anderson-Darling test described by Janacek [47] for determining whether sample data came from normal distributions. The Anderson-Darling test is considered more powerful than the Kolomogorov-Smirnov test and the Ryan-Joiner test [47] described by Janacek which are normality tests that also were considered. The Levene’s test and Bartlett’s test, for instance described by Tjen-Sien and Wei-Yin [95], were used for testing equal variances in two data sets. The two-sample t-test and Welch’s test described by Reed [80] were used for determining whether means of two data sets were statistically different from each other. This approach was useful since the purpose was to investigate if any trends could be seen whether the team improved its results when introduced to the agile process compared to when not introduced to the process. Thus, sample data was collected throughout the study and compared to historical data when the team did not use the process. The first set contained data points when not using the agile process and the second set data when using it. The null hypothesis was that there was no significant discrepancy between the means from the two data sets.

When measuring trends in the data for the metrics related to XP, linear regression analysis was performed. If the regression coefficient in a regression test was larger than 0, it was possible that a discrepancy in the data existed. To validate this, a 95% confidence interval was calculated for the regression coefficient, and if the interval did not contain the value 0 there would be sufficient evidence that there was a discrepancy in the results. This approach to validate if there exists discrepancy in the data assumes a fixed value on the standard error. That is, when calculating a confidence interval one value for the standard error was used, but when the value of the regression coefficient changes the standard error also changes, which this approach fails to address. It should vary as it correlates to the regression coefficient but the approach is assumed to be a good approximation.

Data Presentation. Measurement results were presented in graphs and updated after a completed iteration. Thus, the team received feedback and encouragement for its efforts in the study. In addition to presenting graphs for an iteration, the team received data analysis documents where positive and negative aspects were emphasized. Thus, the team was informed of its software development performance.
9.3.3 Selection of Study Participants

The agile process was introduced to a software development team at the case company. The team worked solely against a single software system and was interested in learning XP. These aspects made the team suitable to conduct research on. Thus, the team was chosen to participate in the study.

9.3.4 Selection of Metrics

Five metrics were used in the study. These were *Project Velocity*, *Average PI Time* (PI is an abbreviation for Product Issue, and is a term used at the case company for identifying software problem reports), *Load Factor*, *Planning Game Tasks, Percentage* and *Estimation Error*. The metrics were chosen based on the software development aspects addressed in the study. Table 9.1 presents the metrics. The term task point is mentioned below. It is an abstract unit used in XP, and refers to one hour of work without distractions and disasters.

Table 9.1: Metrics used in the study.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Velocity</td>
<td>Completed task points per iteration.</td>
<td>A productivity metric.</td>
</tr>
<tr>
<td>Average PI Time</td>
<td>Hours to shift PI's status from New to completed statuses.</td>
<td>An efficiency metric.</td>
</tr>
<tr>
<td>Load Factor</td>
<td>Calendar time divided with completed task points per iteration.</td>
<td>Level of disturbance on the engineers.</td>
</tr>
<tr>
<td>Estimation Error</td>
<td>Estimated time spent on tasks divided with calculated time.</td>
<td>Measures goodness of estimates.</td>
</tr>
<tr>
<td>Planning Game Tasks, Percentage</td>
<td>Percentage of completed tasks planned in planning game.</td>
<td>Support phase’s impact on the planning game.</td>
</tr>
</tbody>
</table>

The metric *Project Velocity* included various tasks concerning software development such as writing specifications and so forth. *Project Velocity* is an XP metric mentioned by Beck [10]. The metric *Average PI Time* addressed only programming tasks, thus these two metrics combined provided a good overview of the team’s overall development effort. *Average PI Time* concerns how long time it takes for the team to complete a PI, that is to transfer it from status new to other statuses that implies that the team has completed the PI. It would be preferred if the trend for this metric was that
a higher number of introduced practices meant a reduction in time. *Load Factor* is an XP metric and is mentioned by Beck [10]. It indicates to what degree an XP team can work undisturbed, provided the estimates of tasks points are accurate. For the metric *Estimation Error* the total amount of estimated time the team spent on completed tasks for an iteration is divided with the value of multiplying the number of completed task points with the load factor from the last completed iteration. The metric *Planning Game, Percentage* provides information how many of the completed task in an iteration that originated from the planning game. This is interesting to measure since the main part of the study meant support job for the team and in support many of the tasks came in randomly during an iteration which reduced the importance of the planning game, where tasks are planned for the coming iteration.

### 9.3.5 Validity

Robson [84] mentions the order effect as a threat to research validity where one treatment is undergone first, another second and so forth. It may be that initial results affect later results where other treatments have been introduced. This is called a "carry-over" effect. A solution for dealing with order-effects is something referred to as counter-balancing. That is, different groups are assigned different orders of carrying out treatments. However, the study had only one study group. Further, nine practices were introduced. This made it impossible to apply counter-balancing since the option of performing research on several groups was not available nor was the time and effort required to carry out all the different orders of introducing the XP practices available.

Further, the study was exposed to the time difference aspect that may affect validity of results. For instance, the team did not consist of exactly the same people at the start of the study compared to at the end of the study.

### 9.4 Results

In the case of comparing results from historical data without using the agile process versus using it, data was divided into two data sets where the first set contained data when the team did not use the process, and the second set data when the team used the process. It was investigated if the two data sets were statistically different from each other. Statistical tests indicated that the collected data did come from normal distributions. In one case, the data sets had equal variances. Thus, the two sample t-test was used. In one case, the data sets did not have equal variances. Here, Welch’s test was used. For the two sample t-test and Welch’s test, the sample size must not be the same in the two data sets.
In the case of investigating trends in data when using the agile process, linear regression was performed on the data. The number of completed iterations was correlated to the average values of the measurement results from the metrics. To further validate if there existed any discrepancies in the data, confidence intervals were calculated for the regression coefficient.

A motivated question in quantitative studies is how the number of data samples affect validity of results. Too few samples may provide incorrect results, while too many data points mean waste of time and effort often for a minimal gain. Factors affected by sample size are confidence interval, confidence level, power value, standard deviation and difference. Based on the available sample data, different values on these factors were calculated in order to understand how the validity of the results was affected.

A power value of 0.8 and a 95% confidence interval was chosen, which can be argued as values that provide reasonable assurance that drawn conclusions are correct. Referring to the actual number of sample data available in the study, the differences for the statistical tests to spot was calculated to be 117 for the metric Average PI Time and 0.163 for the metric Estimation Error. Thus, the tests should produce an alpha value below 0.05 to reject the null hypothesis that states that there is no significant difference between the means from two data sets. The calculations were done using a statistics software package. The alpha parameter is also known as the p-value, that is the probability that an observed effect could have been due to natural variation in the data. The smaller the alpha value, the greater the evidence of a change. Considering these values, high validity of the results could not be assured.

Figure 9.1 to 9.3 present the results for the XP-related metrics. Here, trends for the data collected during introduction of the agile process are analyzed using linear regressions and confidence intervals. For these metrics it was not possible to gather data without using the process. Hence, trends for the collected data were analyzed instead. Figure 9.4 and 9.5 present results for the metrics that allowed comparison to historical data. The figures show box plots that summarize and compare the two data sets’ data points. Here, the analysis concerns comparing the team’s performance using the process to its baseline performance when the team did not use agile method. Thus, the data is divided into two data sets and the data sets’ means are compared to investigate any discrepancies in the data. Figure 9.1 presents measurements results for the metric Project Velocity.

The linear regression provided an r-squared value of 0.705. This indicated correlation between the data sets. The regression coefficient was 3.555. The standard error coefficient was 0.664, which generated a 95% confidence interval for the regression coefficient of +2.253 to +4.857 which did not contain the value 0. Thus, it was shown with a statistical significance that there was an increase in productivity at the end of the study compared to at the beginning of the study.
9.4. RESULTS

Figure 9.1: Measurement trend for the metric Project Velocity.

Figure 9.2: Measurement trend for the metric Load Factor. A low value is preferred since it indicates that the team can work undisturbed. That is, preferably the graph should point downwards.

The linear regression provided an r-squared value of 0.321, indicating little correlation between the data sets. The absolute value of the regression coefficient was 0.190 and since the value was larger than 0, it was possible that a discrepancy in the data existed. The standard error coefficient was 0.080. This generated a 95% confidence interval for the regression coefficient of $+0.033$ to $+0.347$. The interval did not contain the value 0. Thus, it was sufficient evidence that the team could work more undisturbed at the end of the study compared to at the beginning of the study.
Figure 9.3 presents the trend for the percentage of completed tasks originating from the planning game. The blue columns refer to the number of completed tasks that belong to the planning game. The red columns refer to the total number of completed tasks. The reason these columns differ is that tasks arrive during iterations since the team worked mainly in a support and maintenance development environment.

Figure 9.3: Percentage of completed tasks originating from planning games.

For the most part of the study, the agile method was applied in a support phase. That meant minimal development of new functionality. Thus, the team spent its time on solving incoming software problems reported from customers. Often, these tasks arrived during iterations. As a consequence, tasks listed in the planning game were not always completed first since incoming support tasks often had higher priority. Thus, it was interesting to record how many completed tasks originated from the planning game, in order to realize how the support phase affected the planning game. The first iterations did contain new development, but after iteration four it was mainly support phase. After realizing this, measurements from iteration 8 and forward were taken. It was interesting to see that on average only about 49% of total number of completed tasks did originate from the planning game.

Figure 9.4 presents measurement results for the metric *Average Time PI*. The figure shows a box plot for comparing data points from the historical data set when the team did not use the agile method, to data when the team used the agile method. The output from the two sample t-test was a mean of 51 for the first data set, and a mean of 178 for the second data set. Further, the p-value was 0.0 with an alpha level of 0.05. Thus the null hypothesis, that the means from the two data sets were equal, was rejected at a significance level of 0.05. The conclusion was a significant difference in
9.5 Interpreting the results

The case study addressed what impact the agile method had on the software development process in a complex software development environment. Below, analyzes and comments are provided to the measurement results presented in Section 9.4.

For the next 3 metrics mentioned below, there were no comparison to historical data. Instead, trends in the data were analyzed when the agile process was introduced to the team.

**Project Velocity.** For the metric *Project Velocity* there was a significant difference in the data. At the end of the study the team had higher measurements for the metric, compared to at the beginning of the study. Thus, the results indicate that the introduction of the agile process increased
the team's productivity.

Figure 9.1 indicated a trend for increased productivity the longer time the process had been applied. However, the main part of the study concerned support by the team to customers. This meant incoming support tasks during iterations, which the team did not log at the start of the study. Hence, in the beginning many tasks that were actually completed did not receive task points. Thus, the perceived increase in productivity may not be a result of an improved process, but could be the result of a more detailed administration.

**Load Factor.** For the metric *Load Factor* it was concluded that there was a significant difference in the data. However, the confidence interval was just barely above 0. Thus, we should be careful to state that a strong correlation between introducing the agile process and a more undisturbed team exists. However, the results indicate that the team could work more undisturbed after introducing the process.

In the XP literature, it is advocated to start with a load factor of 3 in the beginning of an XP project. Thus, the team started with the value 3. Further, the team deemed 3 was an appropriate number judging how it interacted with the rest of the organization. From iteration four to eight the value increased on average. This was an effect of applying the agile method in a support phase, since the team received many incoming support tasks during these iterations and these tasks did not always receive task points.
However, after iteration nine the team started giving all incoming support tasks points. As a consequence, the value decreased as the graph shows since the Load Factor is calculated as real calendar time divided by task points.

**Planning Game Tasks, Percentage.** For the metric Planning Game Tasks, Percentage the measurement results show that only about half of the originally planned tasks in the planning game was actually completed in an iteration. The rest of the tasks were incoming support tasks. This could both be the reaction of the study being conducted in a support phase, but also due to a large and complex organization which maintains a number of systems. In such an organization, it is natural to have a certain amount of incoming support tasks at a regular basis.

For the metrics mentioned below, there were comparisons with agile data to historical data (i.e., when the agile process had not been introduced).

**Average PI Time.** The comparison of historical data when the agile process had not been introduced to collected data when it was introduced showed for the metric Average Time PI that there was a significant difference in the means for the two data sets, with a higher mean in the second data set. That is, the team actually spent more time solving a product issue when introduced to the process compared to the time it took when not using the process. The standard deviation for the sample data was 198, which was much. This affected the ability to conclude that there was a discrepancy in time for the two data sets. Further, some of the PI’s were created before the study began. Thus, for these PI’s a proportion of the time spent solving them was done before the agile method was introduced. Hence, that data was not affected by introducing the process. However, removing these data points from the study would generate too few data samples for the first data set. Thus they were included as sample data.

The results can be seen as a contradiction since the metrics Project Velocity and Load Factor presented improvements regarding the team’s performance. However, the study took place in a support phase which meant that there was a constant flow of incoming support tasks which needed instant attention. Thus, instead of solving reported product issues, the team solved incoming support tasks. This could explain a reduced efficiency when solving product issues. Thus, an increase in time for solving software problem reports may not mean that the introduction of the agile process had a negative impact on the team’s software development results. Instead, the increased value could be a result of the team prioritizing work on support tasks than on solving software problem reports. However, the data sets’ standard deviations were large and the sample data was limited. This meant a low power value and a rather high p-value. Thus, the result has low validity.

**Estimation Error.** The preferred value of the metric Estimation Error is near one. That means that the estimated time for completing certain tasks is close to the actual amount of time needed to complete these tasks. As Figure 9.5 informs us there was no obvious difference in means for the
two data sets. The comparison of historical data when the agile process had not been introduced to collected data when it was introduced showed for the metric *Estimation Error* that there was no significant difference in the means for the two data sets.

It should be mentioned that for the historical data it were project managers which did not work in the team that made the estimations, but for the collected data when the process was used it were the software engineers who made the estimations. The average mean for the historical data was 1.691, and for the collected data when the process was introduced it was 1.498. It is interesting to see that when the engineers made their own estimations, they tended to be more accurate.

Further, for the most part of the study it was a support phase and support tasks were more difficult to estimate. It could be seen that the estimation error started to fluctuate from iteration five and forward when the support phase was more prominent. This was an effect of applying the agile method in a support phase. Tasks presented to the team of a support character were often vaguely specified. However, from iteration nine and forward the team started to use a phase called pre-analysis. In this phase, the developer analyzed the problem and tried to realize what needed to be done in order to solve the problem. After completing this phase, the developer made his/her estimation and for the later iterations the gathered data show that the measurements were nearer one. Thus, it seemed that applying a pre-analysis phase provided better estimations.

### 9.6 Main Conclusions

Below, the main conclusions from the study are listed.

- The team did not improve its software development results when comparing measurements when using the agile process to when not using the process.

- The team improved its software development results when measuring use of the agile process.

The two conclusions do not contradict each other. The first conclusion compares historical data when the team did not use the agile process to when it did use the agile method. Here, the results did not suggest any improvements when using the process versus not using it. The second conclusion concerns the team when it used the agile method. Here, the data trend showed an improvement of the results.
9.7 Concluding Remarks

This section contains a discussion on the research results and identifies a number of reasons that affected the limited success of the agile method in the study.

To summarize, there was not sufficient evidence to conclude that the team improved its software development results when using the agile process. However, for the metrics related to XP where only trends in the collected data was analyzed, the results showed an improvement in the team’s performance. On the other hand when comparing agile data to historical data when the agile process was not introduced no improvements could be concluded. However, it should be noted that the team did not priority solving PI’s from the database. Instead, focus lay on addressing incoming support issues. That could explain why the team did not improve measurements for the metric Average Time PI when using the agile method whereas the team did improve when analyzing results from measurements for the metric Project Velocity when using the agile process. To summarize when reviewing the results objectively, the null hypothesis that there would be no difference in results when using the agile process compared to when not using it was not rejected.

Although Study 4.1 is part of a case study with limited possibilities to generalize the results to a broader community, the results do not indicate measurable improvements when introducing an agile process based on XP to a complex software development environment. In the case of introducing the agile process to the case company, the reasons for this were several. First, management did not provide enough time and resources for the team to introduce the agile method completely. Another aspect that hindered the success of introducing the process was that the team maintained legacy code which it was reluctant to make changes in. Thus, the practices Refactoring and Coding standard were not applied when maintaining legacy code. For the main part of the study, the team worked with support. The support tasks were often vaguely described, which made it difficult for the software engineers to provide accurate estimations.

For a process based on XP to be successfully applied in these complex software development environments, management and the rest of the organization ought to allow time and resources to adapt the XP practices properly even if it initially may collide with their work routines or interests. Hence, a short term view is not advocated when introducing processes as it takes time to gain the benefits of introducing or adapting a process to a new context or situation. Thus, time is of importance when planning SPI initiatives such as introducing a software development process into an organization.
Chapter 10

STUDY 4.2: QUALITATIVE MEASURES ON AGILE DEVELOPMENT

10.1 Background and Problem

Many software-developing organizations have investigated agile methods as a source for software process improvement. It has been found that agile methods, with most focus on XP (e.g., [10]), provide an effective process that can result in high quality products. However, the typical project where XP is used is rather small with less than 20 developers and it has mostly been used in projects starting without a large legacy base. In the ideal case the change from traditional development should be carried out when a new project of suitable size starts. This means that the development project has a large degree of control and that the team is able to change according to the new working routines at once. The project could be preceded by a training phase where the new routines are learnt, in order to facilitate the change. Further, the organization should be willing to take the risk of making large changes instead of gradually introducing the changes. However, the majority of industrial projects are building on already developed code, the project often interact with a number of instances of the organization, and it is in many cases hard to change too much of the process at once. This means that there is a need to investigate how agile methods could be introduced in these situations as well. In this study, experience is extracted from introducing agile methods in a project working with maintenance of an already delivered product. The studied project works with all types of maintenance, i.e. corrective, adaptive and perfective maintenance.

This study addresses a team’s experiences regarding how a process based on XP was introduced and adapted. The introduced process is presented in Chapter 8. Further, the study investigates how an evolutionary and
maintenance software development environment affects the introduction and adaptation of XP’s practices. The author of this thesis followed the team through the study with interviews and observations. The results are based on interviews and observations conducted on the XP team.

The organization of the study is as follows. Section 10.2 contains related work and section 10.3 presents the research methodology. In section 10.4, the research results are presented. Section 10.5 summarizes the main conclusions from the study and section 10.6 includes concluding remarks.

10.2 Related Works

Agile processes advocate a more informal approach to software development than more traditionally approaches such as the waterfall model described by Royce [85]. By being more informal, they are better suited for handling changes in requirements during development. Processes commonly associated to have an agile approach are XP, lean development [77] and scrum [89], but there are others. XP is the most well known process in the agile family. It has received substantial interest from the software development community with books and research reports written on the subject, as a result. Although the processes have existed for a few years, there is a need for an improved understanding of introducing these processes to organizations as stated for instance by Cohn and Ford [21].

Cohn and Ford [21] present experiences from introducing agile processes into organizations. The experiences are based on introducing XP and Scrum described by Rising and Janoff [82] to different organizations. The paper stresses the fact that introducing an agile process does not only affect the development team members, but also other teams, departments and management. Thus, to succeed with an introduction concerns should be directed to different parts of the organization. Software development approaches that are based on the waterfall model focus more clearly on providing insight in projects. Thus, some managers may be skeptical to agile processes. However, to persuade management to allow agile processes Cohn and Ford [21] suggest creating a model for status reports on projects. The model can include a list of key dates, a short summary of the state of a project and some key metrics. That way, fast and concise status reports can be given to management even if the team develops software in an agile way. Further, Cohn and Ford recognize that introduction of an agile process has impact on other teams. It is important that teams work together in such a way that the teams benefit, or are at least not hindered, in their collaboration. For example, if one team develops software with a heavyweight process and another team develops software in an agile way, there can be communication problems. Cohn and Ford [21] state that it is important that management understand and agree on how introduction of an agile process...
10.3. RESEARCH METHODOLOGY

will impact teams outside the development team. Further, Cohn and Ford argue that organizations should resist distributed development at least for a couple of months after initiating an agile process. Organizations involved in distributed development need to resolve their political and cultural issues before developers can succeed with distributed development. Further, teams involved in distributed development should work together geographically for the first couple of weeks, as a mutual understanding of the development efforts is gained which increases the project’s likelihood of success.

Most of the reported successful applications of XP have involved relatively small projects, typically conducted with a single team. However, Pine [75] describes use of XP in a large project containing 20 developers organized into three teams. The problem was to apply XP in a multi-team environment, where the teams used different software development methodologies and progressed at different speeds. One major obstacle for the teams was integrating systems. The teams formed so called integration pairs, where two people from different teams paired up to discuss integration issues. Integration issues were documented in story cards and planned in integration iterations. In the iterations standard XP practices were followed such as Pair programming and Continuous testing. The results of the project indicated that XP can be successfully applied to large projects that involve multiple teams and different development methodologies.

10.3 Research Methodology

The purpose of the study is to investigate results of introducing an agile process based on XP to an evolutionary and maintenance environment in the industry. To investigate results of introducing the agile process, interviews and observations were made on the software development team which was introduced to the agile process. Analyzes were conducted on the collected data. Based on the results from the analyzes, conclusions regarding the introduction of the agile process were drawn. The team also did subjective rankings of the process’ contents. The purpose of the rankings was to understand how the team perceived the process’ positive effect on development and ease of introduction. The research question is formulated as:

What are the experiences of introducing an agile process based on XP to an evolutionary and maintenance software development environment?

In qualitative studies, conclusions are based partly on the researcher’s assumptions as mentioned by Lantz [58] and Robson [84]. Thus, the assumptions taken in the thesis are presented. It is assumed that introducing agile software processes in evolutionary and maintenance environments is
more challenging than introducing agile processes in less complex environments. It is assumed that XP practices need to be redesigned different in an evolutionary and maintenance environment, compared to the adaptation procedure reported by Beck [10], where the ideal XP environment is presented. It is assumed that the agile process is appreciated and well received by the software engineers in the team.

A flexible qualitative design approach described by Robson [84] was used for the research. Flexible means an evolving design which starts with a single idea or problem that the researcher seeks to understand. The study was explorative in character since results of introducing an agile process to an evolutionary and maintenance software development environment was investigated, which we had limited experience of. The procedure of drawing conclusions from the interviews and observations followed the steps in the model described by Lantz [58] shown in Figure 10.1.

The steps are described in detail in section 10.3.2. The procedure of performing the subjective rankings of the process’ contents is described in Section 10.3.3.

10.3.1 Selection of Study Participants

The first step was to choose study participants. Through contacts at Company X a software development team was chosen to receive the agile process. The team fulfilled the case study’s research needs. That is, a software development team working solely against a system in an evolutionary and maintenance environment. Therefore, when choosing study participants, a non-probability sampling form was used called purposive sampling described by Robson [84]. In purposive sampling selection is made by human choice rather than at random.
10.3.2 Qualitative analysis

Below, the steps mentioned in Figure 10.1 are described in a detailed manner in order for the reader to better understand how the procedure for drawing conclusions from the interviews and observations was conducted.

**Data Collection.** Data was collected through interviews and observations. The participants were interviewed every two weeks. The length of the interviews ranged from approximately 10 to 30 minutes. The higher number of introduced practices, the longer the interview since it was needed to cover usage aspects of every introduced practice. Observations were made at a daily basis. It could be anything from participating unobtrusively in meetings or observing the team while developing software. Below, the methods for data collection are presented.

For the interviews the number of people interviewed at the start of the study were seven. However, during the study two persons left the company. Thus, at the end of the study there were five people left. The interviews were of the type open-guided interviews described by Lantz [58] and Robson [84]. An open-guided interview means a wide research question being enlightened by in advance planned question areas. Here, the interviewees can respond with open answers. That is, respond in such a manner they feel appropriate. This type of interview technique allowed us to be flexible regarding adjusting research questions during the study. Audio software was used to record the interviews. The interviews were not stopped at any time even if something interrupted the interviews, since the purpose was to avoid reminding the interviewees that the interviews were recorded. This could affect the way the interviewee responded, as mentioned by Robson [84]. One person at a time was interviewed. Thus, personal opinions were not biased by other’s opinions. The observations were of the type unobtrusive observation described by Robson [84]. Unobtrusive observation implies an non-reactive approach to the observations where the researcher does not interfere with the study participants. Unobtrusive observation can be structured, but is more usually unstructured and informal. In the study, the observations were unstructured and informal.

**Data Reduction.** To easier perform the analysis, the amount of data to be analyzed was reduced. Here, it is important not to loose information that concerns the research question. The data reduction procedure was to remove sentences that did not address the research question. Further, although sentences of an emotional character are important, follow-up questions were often needed to better understand what the interviewee really meant. Thus, sentences of an emotional character were removed and their associated descriptive sentences were saved. Emotional character means emotional subjective viewpoints like "I really love it!" or "it is the world’s worst test machine!". In these cases the interviewee was asked why he/she really loved something and why the test machine was the world’s worst.
tences of a descriptive character described cause and effect of a phenomenon. It was deemed that in order to draw conclusions on how the practices were adapted to fit the case environment at Company X, logical and descriptive sentences provided more information than sentences of an emotional character. The recorded interviews were transcribed and reviewed. Data reduction was an iterative process. Any contradictions from an interviewee’s response regarding an interview question were solved by presenting the data to the interviewee and asking for an explanation, to better understand what the interviewee meant.

To illustrate the data reduction process a concrete example from the study is provided. The following text was recorded from an interview with one of the study participants. The text shown is a response on the question how the team tested software.

_We may conduct a test race once per week, but it is not continuous. The different parts of the system are integrated in a complex way and it takes time to test the whole integrated system._

Here, the purpose is to recognize what parts of the text that refer to the research question. For instance, the fact that the systems are integrated in a complex manner and it takes time to execute the tests is important. Thus, sentences are kept that address the research question.

**Data Description.** This step prepares data for analysis where possible conclusions may be drawn. The purpose of categorizing data is to enhance the understanding of a phenomenon studied on a more abstract level. By coding the data without losing its meaning, a better overview of the data is provided. First, the codes used to categorize the sentences were defined. The codes reflected the research question, and were based on answers from the interviews. The codes were both descriptive and interpretative, however mostly interpretative due to the inconsistent and sometimes incoherent flow of words in most answers. Thus, each response’s content was interpreted to a certain question, and characterized its content on a more abstract level. The method used to categorize the sentences was to analyze every sentence and note which code the sentence received. If the categorization was ambiguous, it was marked with a certain notation. If several ambiguous categorizations were received this indicated that the codes were unclear and needed a clearer definition regarding their content and scope. Thus, the data was characterized on a more abstract level. After coding the data, it was possible to identify if some dimensions were more or less empty. This indicated that some dimensions had been described too little. Hence, in following interviews these dimensions were probed into more, to better understand the phenomenon.

To illustrate the data description process an example from the study is provided. The text shown is a response on a question if there were any problems associated with making changes in legacy systems.
When making changes in legacy code, there is a risk that you mess things up so the code does not execute as planned, and that risk is something you want to avoid as much as possible.

The sentence was replaced by the code "Changes in legacy code are preferably avoided due to fear of inserting defects in the code." Thus, the text is described in a manner that better answers the research question. It is possible to classify several answers to the same code even though they differ in wording.

**Pattern Finding.** The purpose of this step is to gather data, enabling reflection of how codes correlate to each other and the research question. By seeking patterns in the data, it is possible to draw conclusions that answer the research question. The data can be ordered in a number of ways. The data was ordered in tables. The columns were the dimensions of the research question. The dimensions were defined by analyzing the interview answers and relating these answers to the research question. This way, the author of this thesis could realize that the answers concerned certain parts of the research question. These parts became the dimensions. Thus, the dimensions reflected the contents of the answers. By dividing the research question into dimensions it was possible to better understand cause and effect for certain parts of the research question. The rows consisted of answers from the study participants, categorized into codes. This is referred to as a conceptually clustered matrix by Robson [84]. The matrix form was chosen since it provided a good overview of how codes correlated to each other when trying to answer the research question. However, it is important to understand that other groupings of data may lead to different emerging patterns and conclusions. Thus, seeking patterns is an iterative process.

To illustrate how data can be ordered when seeking patterns an example is provided from the study. Table 10.1 presents two aspects of the research question. Each row in the table corresponds to codes from one study participant on experiences of implementing the practice Continuous integration in the case environment. By viewing codes ordered in some form, the researcher can find patterns which may provide a deeper understanding of the phenomenon studied.

**Critical Review.** The last step of the qualitative data workup process, presented in Figure 10.1, is a critical review of the drawn conclusions. The purpose is to investigate if alternative interpretations and conclusions can be drawn. Thus, the validity of the conclusions is tried. In the study, a fellow research colleague repeated the data workup step Pattern Finding. The colleague was familiar with XP and the purpose of the study. Hence, he had the same conceptual frame of reference, which is needed to provide valid conclusions as stated by Lantz [58]. If there were major differences between our and the colleague’s conclusions we discussed why this was the case. Based on the discussion it was decided which conclusions were most
Table 10.1: The team’s codes for experiences of implementing the practice *Continuous integration*.

<table>
<thead>
<tr>
<th>The case environment’s influence on the implementation of the practice <em>Continuous integration</em></th>
<th>The organization’s influence on the implementation of the practice <em>Continuous integration</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Certain support issues are not applicable to continuous integration.&quot;</td>
<td>&quot;Projects with tight deadlines may not approve continuous integration activities if these take time to perform.”</td>
</tr>
<tr>
<td>&quot;The case environment facilitates use of continuous integration.&quot;</td>
<td>&quot;Parts of the practice are already applied.”</td>
</tr>
</tbody>
</table>

valid. However, there were no major differences in our conclusions. In some cases though, we arrived at different conclusions but they did not contradict each other. Further, the drawn conclusions were presented to the study participants to get their feedback and better determine the validity of the results. This verification approach is referred to by Robson [84] as getting feedback from informants. The conclusions were shown individually to each participant. This way, personal opinions were not biased by other’s opinions. The results based on the analyzes of the interviews and observations are presented in Section 10.4.

### 10.3.3 Subjective Rankings

The team was asked to subjectively rank how easy it was to introduce the process’ contents, and how much effect the introduction of the practices had on the team’s performance. In order to carry out the ranking a method based on the Analytical Hierarchy Process (AHP) described by Saaty [87] was used. The idea of the method is that it is easier to compare objects pairwise than to directly rank all objects. The method assumes $n$ objects as input and requires that $n(n - 1)/2$ comparisons are made, i.e. every possible pair of objects are compared. The AHP method has been applied previously in a software engineering context, see e.g. the works by Karlsson et al. [53] and Karlström [54]. The AHP method is not presented in detail in this study. This information can, for example, be found in [87] or [53].

The original AHP method was intended to be used once by all participants together, i.e. for every pairwise comparison all participants should agree on one value. The result of the approach is one vector $p = (p_1, p_2, \ldots, p_n)$ representing the relative weight of every object, where $\sum_{i=1}^{n} p_i = 1$. In this case the participants carried out the comparisons individually, i.e. the approach results in individual vectors. In this case the result is one vector
10.3. RESEARCH METHODOLOGY

from every person representing how easy it is to introduce the practices and one vector from every person representing how much effect the use of XP’s practices had on the team’s performance. In order to get one vector representing how easy it is to introduce the practices and one vector representing the effects, the individual comparisons must be aggregated in some way. Forman and K. Peniwati [33] present a number of approaches for this and in this case the elements of the vector was calculated as the (arithmetic) mean of the elements of the individual vectors. That is, the resulting element $p_i$ was calculated as the mean of the individual $p_i$-values.

Since the method involves many comparisons it is possible to estimate the consistency in the form of a consistency ratio of every person. A consistency ratio of $0.10$ or less is considered acceptable, but e.g. according to Karlsson and Ryan [53] consistency ratios in practice exceed $0.10$ frequently.

10.3.4 Validity

Robson [84] mentions the order effect, where one treatment is undergone first, then another and so forth. It may be that initial results affect later results where other treatments have been introduced. This is called a ”carry-over” effect. A solution for dealing with order-effects is counter-balancing. That is, different groups are assigned different orders of carrying out treatments. However, in the study, there was only one group. Further, only nine practices were introduced. This made it impossible to apply counter-balancing since we did not have several groups neither the time required to carry out all orders of introducing the practices.

A number of threats to flexible design studies that applied to the study are mentioned by Robson [84]. Description, respondent biases and researcher biases are all threats that may affect the validity of results gained. Description refers to the risk of providing invalid descriptions of what you have seen or heard, based on inaccuracy or incompleteness of data. To counter this, audit trail described by Robson [84] was applied. Interviews were recorded with a lap top with audio software, transcribed and stored electronically. Further, daily notations were written down to maintain a research log. Respondent bias refers to how respondents interact with the researcher, for instance providing answers they judge the researcher wants. To counter this, it was made clear to the respondents that the research was neutral to whether XP was considered a success or failure, thus they did not have to feel any pressure for liking or disliking XP. Researcher bias concerns how the researcher’s assumptions and preconceptions may affect the researcher’s behavior in the study, perhaps in terms of which people are selected for interviews, type of questions asked and selection of data for analysis. To counter this, peer debriefing [84] was applied in order to get feedback from research colleagues that may help us realize our own, maybe, somewhat faulty assumptions. Peer debriefing is talking with a trusted colleague familiar in the
research area about our experiences and assumptions in the research, and receiving feedback.

When people know they are under observation, they may behave differently due to the fact that they are under observation. This is called the Hawthorne-effect, mentioned by Rumpe [86]. This is a validity threat to the study. To reduce this threat a non-participatory approach was applied when observing the study participants. A non-participatory observation approach means observing study participants in a non-obtrusive way, thus the researcher does not participate in the study. Further, the fact that the study took place under real world conditions in the industry probably made the study participants focus on their daily work instead on the study, which may have reduced the Hawthorne-effect.

It should also be noted that the results are based on interviews and observations with and on a limited number of people from one single company. Thus, the results do not permit statistical generalization concerning introducing XP to evolutionary and maintenance software development environments in general.

10.4 Results

The introduction of an agile process affects more people in an organization than just the software engineers, as confirmed by Murro et al. [67]. The results presented in this study are based on the team’s perspective concerning the introduction of the agile process. The organization’s perspective regarding the introduction of the agile process is addressed in Study 4.3.

The interview answers and the observations underwent a qualitative analysis, as described in Section 10.3.2. Based on the analysis conclusions were drawn concerning introducing the agile process in the company’s software development environment. The conclusions based on the interviews are presented in Section 10.4.1. The conclusions based on the observations are presented in Section 10.4.2. After introducing the agile process, the team was asked to rate the practices according to their positive effect on development and their ease of introduction. The results of the rankings are presented in Section 10.4.3.

10.4.1 Results from Interviews

First, conclusions regarding the overall experience of introducing the agile process at the case company are provided. Then, experiences regarding XP’s practices are discussed. The conclusions regarding the overall experience of introducing the agile process at the case company are presented below.

1. No pattern can be seen regarding different roles in XP and opinions when introducing the agile process. The coach and the programmers
10.4. RESULTS

mentioned the same problems and opportunities when introducing the process.

2. The case environment facilitates use of code related practices, since the team maintained and supported systems in a complex environment.

3. The case environment complicates introduction of non-code related practices (e.g. the practice Planning game), since team members participated in different projects which made it difficult to work in pairs. Further, incoming support issues made planning difficult.

4. Prior to introducing the agile process, an assessment of the organization’s software development processes should be done, to understand what should be introduced and why.

5. Introducing an agile process based on XP into the case environment was believed to be more difficult than to an ideal environment. Organizational and cultural issues made it difficult to introduce the process as XP advocates.

The study provides experiences regarding the introduction of XP’s practices, based on interviews with and observations on the team. The case environment facilitates use of the practice Collective code ownership. The team often worked with finding and resolving defects, and when locating defects the software engineers must often review several systems. The case environment do not facilitate use of the practice Continuous integration. Many tasks were of a support character. For these, a large part of the time is spent on analysis and a small part on coding. Thus, there was seldom integration of code. Regarding the practice On-site customer, it was noticed that the case environment means that customers tend to prioritize between software problem reports instead of discussing different possible solutions on solving these defects with the team. The case environment may mean several customers to the team, and several customers may cause prioritization problems for the team regarding which customer should be attended to next. The case environment means large efforts to introduce the practice Continuous testing completely. The existence of many languages, platforms and dependencies between systems makes it difficult to create test frameworks for the whole environment. The team perceived that the case environment complicates use of the practice Simple design. Dependencies between systems require advance planning when performing design activities. The practice Planning game made the developers more aware and knowledgeable about typical project management issues.
10.4.2 Results from Observations

As mentioned in Section 10.3.2, the observations were of an informal and none participatory kind. Thus, the study participants were not disturbed when observed. Further, pen and paper were used to record the observations for later analysis. The observations confirmed conclusion number two and the experiences made from the interviews. In addition to that, the following conclusions were made based on the observations.

1. Introduction of the agile process facilitates transition of knowledge, since the process is based on communication.

2. Lack of management support complicates introduction of the agile process, since it requires time and equipment to introduce the process properly.

3. The case environment raises the need to clarify XP terminology. The team collaborates with its organization and needs to translate others’ terminology to its own.

The study provided experiences regarding introducing XP’s practices, based on observations of the team. The experiences from the observations were the same as for the interviews. The experience for the practice Continuous integration is the same as described in Section 10.4.1, which was based on conducted interviews. The experience for the practice On-site customer is the same as described in Section 10.4.1, which was based on conducted interviews. The practice Planning game is not suited for the case environment. Often, there was a continuous stream of incoming support tasks with high priorities. Thus the initial tasks at the planning game were seldom completed. The support tasks were more difficult to estimate as the engineers often had minimal information regarding these tasks. Thus, the reasons for software failures were mostly unclear and it often involved complex search routines to localize causes for the problems. Further, several customers mean prioritization problems. The experience for the practice Continuous testing is the same as described in Section 10.4.1, which was based on conducted interviews. The experience for the practice Coding standard is the same as described in Section 10.4.1, which was based on conducted interviews.

10.4.3 Results from Subjective Rankings

The subjective rankings of the practices followed the steps of the Analytic Hierarchy Process, AHP, as described in Section 10.3.3. The ratings were consistent thus having high credibility.

Figure 10.2 presents rating results for how the team perceived the positive effect on development each practice had. Maximum value is 1, which means 100 percent, and minimum value is 0.
10.4. RESULTS

Figure 10.2: Rating results for practices concerning positive effect on development.

The practices were perceived to contribute evenly to positive effect on development. It is interesting to notice that the practice On-site customer received the lowest value, even if the practice meant some changes in how the team worked. However, there was no real on-site customer present and the customer representatives whom acted the on-site customer role were not present all the time and did not belong to the team. The practice Coding standard received the highest value. This result may depend on the fact that the team prior to introducing the agile process already used coding standards, so the team was familiar with the concept and appreciated its value. Thus, the practice was well received.

Figure 10.3 presents rating results for ease of introduction of the practices. The higher value a practice receives, the easier it is to introduce according to the team. Here, the ease of introduction concerns the introduction of the adapted practices, and not the ease associated with introducing the standard XP. This is important to recognize since many of the practices were adapted and therefore it is their ease of introduction that should be considered.

Here, there was a wider range how the team perceived the practices’ ease of introduction. The practice Simple design received nearly twice as many percent as the practice Continuous testing. The practice Simple design did not mean any differences in how the team worked, thus it had little impact. As a result, the practice was considered easy to introduce. The practice Continuous testing received the lowest value which was no surprise, since that practice is reported in other works to be difficult to introduce which the team also experienced. It took time and effort to install the unit test framework. The team also worked on several platforms with different
operating systems which meant additional work to install test frameworks on all platforms. Thus, the practice was considered difficult to introduce. The practice *Pair programming* was also considered difficult to introduce. This depended partly on the fact that there was mostly support during the study and the team did not appreciate sitting in pairs when solving support issues. Another fact that made the practice hard to introduce was that some team members did not, under any circumstances, appreciate sitting in pairs while programming.

10.5 Main Conclusions

Below, main conclusions from the study are listed. The main conclusions are subjectively derived and describe, in our opinion, important conclusions regarding introducing an agile process to an evolutionary and maintenance software development environment. They are based on the results presented in Section 10.4.

- Prior to introducing an agile process, an assessment of the organization’s software development processes should be done to understand what should be introduced and why.

- The case environment raises the need to clarify XP terminology. The team collaborates with its organization and needs to translate others’ terminology to its own.

- Introducing an agile process based on XP into the case environment was believed to be more difficult than to an ideal environment. Orga-
izational and cultural issues made it hard to introduce the process as the standard XP.

- Introduction of the agile process facilitates transition of knowledge, since the process is based on communication.

- Use of the practice *Planning game* makes developers more aware and knowledgeable about typical project management issues.

- The case environment means large efforts to introduce the practice *Continuous testing* completely. The existence of many languages, platforms and dependencies between systems makes it difficult to create test frameworks for the whole environment.

- The case environment facilitates use of code related practices, since the team maintained and supported systems in a complex environment.

The conclusions indicate that the case environment facilitates use of code related practices. This conclusion may be explained as a result of maintaining and supporting several systems. When doing so, the software engineers may benefit of having an ordered structure in their work which these practices offer. This opinion is supported by the results from the subjective rankings done with the AHP method, presented in Section 10.3.3. The team perceived that the three practices with the most positive effect on development were code related practices. Further, they also ranked code related practices to be the most difficult to introduce. Thus, providing support to introduce these practices seems important to improve the software development process. As stated above, it requires large efforts to introduce the practice *Continuous testing* in an evolutionary and maintenance software development environment. However, it should be stressed that it is not necessary to introduce the practice early, nor apply it only on new code. Research provided for instance by Zhang [99] show positive results when applying continuous testing on legacy code.

The empirical data presented in this study is based on subjective opinions of those interviewed. However, to better understand the effects of introducing the agile process, the quantitative data presented in Study 4.1 should be considered. When comparing productivity results before versus after the agile process was introduced, no significant improvement could be seen. However, as Figure 10.4 shows, there was an improvement of productivity during the introduction of the agile process. Project velocity is a productivity metric for measuring XP performance, described by Beck [10].

Thus, there is evidence that when the agile process was introduced the team improved its productivity. Hence, we can argue that the introduction of the agile process had a positive effect when referring to the analysis of the quantitative data. This conclusion supports the views of the study...
participants as they argued that the case environment facilitates use of code related practices. A heavy use of code related practices is likely to have a positive effect on the team’s productivity. Further, the team was of the opinion that the introduction of the agile process facilitates transition of knowledge. An improved level of knowledge in the team members is also likely to have a positive effect on the team’s productivity.

10.6 Concluding Remarks

The purpose of this study is to investigate results of introducing an agile method based on XP, into an evolutionary and maintenance software development environment. The conclusions were based on interviews and observations of a team that started using the agile process. The results from the interviews and observations were analyzed, and conclusions based on the analyzes were drawn. Further, the team performed pairwise comparisons of the process’ practices with regard to positive effect on development and ease of introduction. The study may help other organizations with similar software development environments to introduce agile processes based on XP. The conclusions drawn indicate that it is more difficult to introduce XP, in its original appearance, to an evolutionary and maintenance software development environment than to less complex environments as described by Beck [10].

One important conclusion drawn from the interview results was that it is important to assess existing development processes before introducing an agile process into an organization, to realize what to introduce and how.

By observing the team adapting the XP practices, an important conclusion was drawn which is that an evolutionary and maintenance software
development environment raises the need to clarify XP terminology. The team interacted with many people in the organization and was involved in a number of projects. Many of these contact interfaces had their own ways of reporting time, defects and so forth which made it important for the team to clarify XP terminology so that the team could translate external project terms to its XP environment.

The results from this study add to the empirical body of knowledge concerning introduction and use of processes based on XP in various contexts, which is needed as for instance stated by Layman et al. [59]. Further, as reported by Sharp and Robinson [90] it is not necessary to strictly follow all practices without modifications in order to create an appropriate culture for developing software in an agile way. Instead, a modified, different set of practices might produce the necessary requirements. These results support the results presented in this study. That is, the introduction of an agile process based on XP can still be successful even though not all of the twelve practices are introduced and those introduced have been adapted to the team’s development environment. Thus, the results from this study show that it is not necessary to apply all of XP’s practices in order to achieve improvements in the software development process. This may seem obvious but is still worth mentioning. It is our impression that parts of the agile community strongly advocates the use of all of XP as the practices are interconnected with each other, and full benefit from XP can only be achieved if all of XP’s practices are applied simultaneously. However, this viewpoint is not supported by facts based on results from empirical studies. On the contrary, as this study shows significant support can be achieved by only applying parts of XP, which in turn have been adapted to a certain extent. The results presented in this study and the results mentioned by Layman et al. [59] and Sharp and Robinson [90] will hopefully spur others to conduct more research in the area.
Chapter 11

STUDY 4.3: AGILE COLLABORATION BETWEEN TEAM AND ORGANIZATION

11.1 Background and Problem

Successful software development projects are characterized by project teams that collectively take responsibility of their engineering processes. Further, a team should recognize the team members’ viewpoints on how team performance can be improved, as stated by Katzenbach et al. [55]. This is also true when developing software, as confirmed by Reifer [81]. Software development projects of today are often large and involve a lot of people with different abilities. It is important to recognize their viewpoints on how collaboration can improve, in order to deliver fault free software on time. The study addresses this issue, by recognizing how an organization perceives its collaboration with a software development team.

Agile methods have received interest from the software development community during the last years, perhaps mainly due to the fact that they address changes which is important in software development. Agile methods emphasize simple and direct communication as a means for improving software development results. Communication is facilitated as people sit close to each other, daily meetings are performed and charts on team performance are made visible. These aspects and more enable a simple and direct communication, which help manage changes in software development. Further, collaboration is based on communication as mentioned by Abrahamsen [4]. Thus, the use of an agile process ought to improve collaboration in an organization, as agile methods facilitate communication. The purpose of the study is to investigate whether this is the case.
In the study, an agile process was introduced to a large organization with a complex software development environment, which included both support and maintenance of several systems. The agile method is presented in Chapter 8. In these kinds of environments changes occur frequently such as new versions of requirements and re-prioritization of tasks. A team applied the agile process during eight months. The team wanted to improve its software development process by using this kind of process, as iterative development and an improved ability to handle changes were appealing to the team. The process was based on XP [10], but not identical to it since some parts of the process were not introduced and others were adapted to fit the team’s development environment. The team collaborated with its surrounding organization to develop software.

This study investigates how collaboration between a team and its surrounding organization was affected, from an organizational point of view, during introduction of the agile process. Most of the research published about agile processes are based on viewpoints from teams applying these processes, but this study provides input on how an organization considers use of an agile process. Hence, interviews were conducted with the organization on how it perceived its collaboration with the team before, during and after the process was introduced. The results of the study may help understand how agile development affects collaboration in organizations, seen from an organizational perspective. This study is part of the case study, presented in Chapter 8, where the purpose is to investigate the effects of introducing an agile process based on XP to a large organization with a complex software development environment. The results presented in this study concern how the surrounding organization perceived that its collaboration with the software development team changed as the agile process was introduced to the team. The results presented in Study 4.1 address the team’s performance by analyzing quantitative data before and after the agile process was introduced. The results in Study 4.2 concern how the team perceived their use and adaptation of the agile process.

Section 11.2 contains related work and section 11.3 presents the research methodology. In section 11.4, the research results are presented. Section 11.5 summarizes the main conclusions from the study and section 11.6 includes concluding remarks.

11.2 Related Works

Most of the research concerning introduction and use of agile processes are based on opinions from the teams applying these processes such as [4], [82], [38] and [88]. However, as stated by Reifer [81] it is important to recognize viewpoints on improvement issues from different perspectives when developing software.
Murro et al. [67] report on an Italian company who introduced XP in two projects. The company had recognized the fact that introducing an agile process into an organization affects more people than just the software engineers. The output from the study were experiences of introducing XP from a non-programmer’s perspective. Three different perspectives were presented including viewpoints from project managers, customers and an quality assurance team. Two observations were made from the customer’s point of view. First, customers tend not to accept requirements which are not written down. The customers representatives felt that it would be difficult getting contracts signed without formally specifying the requirements. The second observation was that people representing the customer did not own the customer’s requirements. This meant that they sometimes lacked the authority or knowledge to resolve issues. Often, they had to contact the real customer and ask for advice. Further, this detachment of a real customer in the team reduced the emotional commitment from the customer. The project managers observed that XP allowed software engineers to maintain and enhance their programming abilities. That is, through practices such as Pair programming, Collective code ownership and Coding standard were the engineers supported to improve their skills in software development. The company was ISO 9001-certified, and the quality team observed that using XP improved the company’s quality system by providing guidelines for project management. Further, the quality team noticed that XP focuses on a few key process aspects which allowed the software engineers freedom to customize the rest of their work processes. Therefore, the quality team concluded that XP can be adapted to fit the ISO 9001 framework.

To summarize, the non-programmer’s perspective on using XP was that XP supports a software development team to improve its development skills. However, it is important to have a real customer in the XP team to handle business issues when developing a system. Further, XP is not a hinder for an organization pursuing a quality management system based on ISO 9001 as its contents can be adapted to the framework.

11.3 Research Methodology

This section presents the research method used when drawing conclusions on how the organization perceived the agile process’ effect on its collaboration with the team. Further, the validity of the study is discussed. The study is conducted at a Swedish software development company, described in Chapter 8.

As stated earlier, collaboration is based on communication as confirmed by Abrahamsen [4]. Further, agile methods facilitate communication in various aspects. For instance, new requirements are addressed through frequent and informal contacts with customers. Another example is the practices Pair
programming and Planning game which are used as means for intensifying communication in software development teams. Based on this argument, the author of this thesis claims that agile development improves collaboration in an organization as agile methods facilitate communication. Thus, the research question is:

*How does the use of an agile process affect the collaboration in a software development organization?*

The term collaboration can mean different things to different people. Therefore, it is motivated to define what the term means in the study. Blomqvist et al. [12] define the concept of collaboration as communicative interaction, where trust is considered a critical success factor for improving collaboration. Trust is gained when mutual goals are achieved and when involved parties can rely on each other that joint activities will be performed as agreed. An increased trust is usually a sign of improved collaboration. In the study, the term collaboration is defined as working together where several parties may achieve something beyond what either would have achieved separately. This concept also involves trust. Thus, an improved collaboration is based on whether the organization perceived that it increased its trust to the team and that they performed better together than they did prior to the introduction of the agile process.

Further, how collaboration is affected due to the introduction of a software development process may be investigated in a number of ways. The approach that was chosen in the study was to interview people from the organization before, during and after the introduction of the process and perform an analysis of the recorded interviews, as described below. This approach had several advantages. First, since there was limited knowledge of how the team and its surrounding organization collaborated, it was difficult to know where on a more detailed level the research should be performed. For instance the collaboration efforts could be modelled in some sort of a communication path, but such approaches are based on an extensive knowledge how the team collaborates with its surrounding organization. Further, since there is little research performed on this issue there is a risk of missing important information if a more detailed research approach is chosen. The interview approach used in the study addresses this issue, since the interviewees provide information from different angles, based on their experiences. Hence, the risk of missing important information was reduced by performing interviews on the organization.

A flexible qualitative design approach [84] was used for the research. Flexible means an evolving design which starts with a single idea or problem that the researcher seeks to understand. The study was explorative in character since it investigated how agile development may affect the collaboration between a software development team and its surrounding organization, which there was limited experience of. The procedure of drawing
conclusions from the interviews followed the steps in the model described by Lantz [58] shown in Figure 11.1.

Below, the steps mentioned in Figure 11.1 are described in a detailed manner in order for the reader to better understand how the procedure for drawing conclusions from the interviews was conducted.

**Data Collection.** Data was collected through interviews. Three participants were interviewed. Each participant was interviewed three times, one time before the agile process was introduced, one time during introduction of the process and finally one time after the process was introduced. The length of the interviews ranged from approximately 10 to 30 minutes. The interviews were of the type open-guided interviews described by Lantz [58]. Open-guided interviews mean a research question being enlightened by in advance planned question areas, where the interviewees can respond with open answers. Audio software was used to record the interviews. One person at a time was interviewed. Thus, personal opinions were not biased by other’s opinions.

**Data Reduction.** To easier perform the analysis, the amount of data to be analyzed was reduced. The data reduction procedure was to remove sentences that did not address the research question. The recorded interviews were transcribed and reviewed.

To illustrate the data reduction process an example is provided from the study. The following text was recorded from an interview with one of the study participants. The text shown is a response on the question how the study participant communicates with the team.

> Often I communicate with the team rather informally. It can be at a lunch or if I meet a team member in the corridor. Personally, I like that approach, formal meetings require preparation and take a lot of time.

Here, the ambition is to recognize what parts of the text that refer to the research question. For instance, the fact that communication is of an informal nature is relevant to the question, thus that sentence is kept. However,
the sentence where the respondent states his/her preference for informal communication does not address the question. Thus, that sentence is not kept.

**Data Description.** This step prepares data for analysis where possible conclusions may be drawn. By coding the data without losing its meaning, a better overview of the data is provided. Each response’s content is interpreted to a certain question, and characterized on a more abstract level in forms of codes. Responses from different interviewees could be classified to the same code, if the meanings of the responses were the same although not necessarily with the same wording.

To illustrate the data description process an example from the study is provided. The text shown is a response on the question how often the respondent communicates with the team.

> Well, I communicate with the team a few times a week if I would make some kind of average guess. It is not every day at least.

The response was replaced by the code "Communication occurs a couple of times a week on average." Thus, the text is described in a manner that better addresses the research question.

**Pattern Finding.** The purpose of this step is to structure gathered data, in forms of codes, thus enabling reflection on how the codes correlate to each other and the research question. By seeking patterns in the data, conclusions may be drawn that answer the research question.

To illustrate how data can be ordered when seeking patterns an example is provided from the study. Table 11.1 presents two aspects of the research question. These aspects divide the research question into subparts which may help draw valid conclusions. The aspects were identified when interviewing the study participants. Each row in the table corresponds to codes from one study participant on experiences how his/her collaboration with the team changed as the agile process was introduced. By viewing codes ordered in some form, patterns can be found which may provide a deeper understanding of the phenomenon studied.

**Critical Review.** The last step of the qualitative data analysis process presented in Figure 11.1, is a critical review of the drawn conclusions. The purpose is to investigate if alternative interpretations and conclusions can be drawn. Thus, the validity of the conclusions is tried. The conclusions were reviewed subjectively by us, in an attempt to validate them.

### 11.3.1 Selection of Study Participants

Customer representatives, which collaborated with the team, acted as intermediaries of the organization’s viewpoint on how the agile process affected the organization’s collaboration with the team. Although the results of the
Table 11.1: Experiences made by the organization when collaborating with the team.

<table>
<thead>
<tr>
<th>Team influence on collaboration</th>
<th>Organizational influence on collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;The team provides better time estimates after the introduction of the agile process.&quot;</td>
<td>&quot;Internal projects require much attention, which reduces available time to communicate with the team.&quot;</td>
</tr>
<tr>
<td>&quot;The team answers questions faster after the introduction of the agile process.&quot;</td>
<td>&quot;The organization does not understand terms used in the agile process, which makes communication with the team more difficult.&quot;</td>
</tr>
</tbody>
</table>

study are based on interviews with customer representatives, it is the organization’s viewpoint that is gathered. The customer representatives had other work tasks than those that were customer related. Thus, they were part of the organization and should not be considered as customers.

The customer representatives were interviewed before, during and after the agile process was introduced. The number of customer representatives was three which is many in terms of number of customers in XP projects. Therefore, considering the nature of the study which concerns organizational input based on customer representatives involved in agile development projects, the number of study participants is adequate. Qualitative analyzes were conducted on the interview results, and conclusions were drawn based on the analyzes. The conclusions drawn before introducing the agile process, were compared with conclusions drawn during and after the process was introduced. Thus, any effects the agile process had on the collaboration between the surrounding organization and the software development team was investigated.

11.3.2 Validity

A number of threats to flexible design studies that applied to the study are mentioned by Robson [84]. Description and respondent biases are threats that may affect the validity of results gained. Description refers to the risk of providing invalid descriptions of what you have seen or heard, based on inaccuracy or incompleteness of data. To counter this, audit trail was used. That is, the interviews were recorded with audio software, transcribed and stored electronically. Further, daily notations were written down to maintain a research log. Respondent bias refers to how respondents interact with the researcher, for instance providing answers they judge the researcher wants. To counter this, it was made clear to the study participants that our researcher role was neutral to whether the introduced process had any positive...
effects on the collaboration or not, thus they did not have to feel any pressure for liking or disliking the process. The countermeasures were subjectively considered to have a positive impact on the validity of the results.

11.4 Results

The interview answers underwent qualitative analyzes according to the steps described in Section 11.3. The customer representatives were interviewed before, during and after the agile process was introduced. Based on these analyzes a number of conclusions were drawn concerning collaboration issues between the team and its surrounding organization. Thus, the study investigated how agile development affected the collaboration between the team and its surrounding organization. The results are based on interviews with the customer representatives. That is, from an organizational point of view.

11.4.1 Changes in Collaboration

The introduced process changed how the team and its surrounding organization collaborated. Below, a summary of the practices that had most impact on how the team and its surrounding organization collaborated is presented. These practices were identified by interviewing people in the organization that collaborated with the software development team.

Planning game. The planning aspect mentioned in the practice *Planning game* was important for the team when collaborating with its surrounding organization. The organization’s current and future needs were included in the team’s planned software development releases. The practice *Planning game* had a positive effect on the collaboration since it provided the organization with a better insight of the team’s software development progress.

Small releases. The team delivered new software releases to the organization, as covered by the practice *Small releases*. To continuously release new versions of software was important in the collaboration between the team and its surrounding organization, since the organization was able to continuously review a system while it was being developed.

On-site customer. The practice *On-site customer* addresses an important aspect that affected the collaboration between the team and its surrounding organization, and that is the role of the customer. The team discussed development issues with the customer representatives. These meetings acted as a communication channel between the team and the organization. The practice *On-site customer* had a positive
effect on the collaboration between the team and its surrounding organization since it provided the organization and the team with means for discussing software development issues on a more detailed level.

The practice 40-hour week addresses an important issue in software development projects and that is time. The standard version of XP does not recommend working overtime more than a couple of weeks in a row. The team applied the practice whenever possible, but if something urgent needed to be solved the team worked overtime if necessary. Thus, the collaboration was not affected negatively seen from an organizational perspective since the team was always available. However, if the practice would have been applied as advocated by the standard XP it is likely that the organization would had regarded this as negative due to the team’s refusal of working overtime.

11.4.2 Conclusions Before Agile Development

Below, conclusions are listed concerning collaboration issues between the team and its customer representatives before introducing the agile process.

1. The customer representatives feel that there is sufficient communication with the software development team.

2. The customer representatives do not trust that the software development team will deliver software when it has promised to.

3. The customer representatives feel that the software development team does not inform them about recent changes in work issues. The team should provide feedback more often to the customer representatives.

4. The customer representatives feel they provide the software development team with enough information so that the team can perform well with their issues.

5. There does not exist any formal way of communicating with the team. For instance, it varies which person in the team is contacted, how often and in which way.

11.4.3 Conclusions During Agile Development

Below, conclusions are listed concerning collaboration issues between the team and its customer representatives during introduction of the agile process.

1. The customer representatives feel that there is sufficient communication with the software development team.
2. Two of three customer representatives feel that the software development team has improved its ability of keeping delivery dates.

3. Two of three customer representatives feel that the software development team reduced the elapsed time before providing the customer representatives with answers.

4. The customer representatives feel that the software development team does not inform them about recent changes in work issues. The team should provide feedback more often to the customer representatives.

5. There does not exist any formal way of communicating with the team. For instance, it varies which person in the team is contacted, how often and in which way.

11.4.4 Conclusions After Agile Development

Below, conclusions are listed concerning collaboration issues between the team and its customer representatives after introducing the agile process.

1. The customer representatives feel that there is sufficient communication with the software development team.

2. Two of three customer representatives feel that the software development team has improved its ability of keeping delivery dates.

3. Two of three customer representatives feel that the software development team reduced the elapsed time before providing them with answers.

4. Two of three customer representatives feel that the software development team informs them about recent changes in work issues.

5. The customer representatives feel that the software development team has improved its competence level and spread it throughout the team.

6. There does not exist any formal way of communicating with the team. For instance, it varies which person in the team is contacted, how often and in which way.

11.4.5 Trends Regarding Collaboration Between Team and Organization

This section presents trends regarding the organization’s perception of how its collaboration with the team changed as the team was introduced to the agile process. The trends regarding the organization’s viewpoints are based on a subjective analysis of the drawn conclusions before, during and after the introduction of the agile process. The trends are shown in Table 11.2.
Table 11.2: Trends regarding collaboration issues between the team and its surrounding organization.

<table>
<thead>
<tr>
<th>Trend</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is sufficient communication between the team and its surrounding organization.</td>
<td>The introduction of the agile process did not affect how often the team and its surrounding organization communicated, according to the organization.</td>
</tr>
<tr>
<td>There does not exist any formal way of communication between the team and its surrounding organization.</td>
<td>The introduction of the agile process did not affect how the team and its surrounding organization communicated, according to the organization.</td>
</tr>
<tr>
<td>The organization feels that the team has improved its ability of keeping delivery dates.</td>
<td>The introduction of the agile process had an improvement of the team's ability to deliver software at agreed dates, according to the organization.</td>
</tr>
<tr>
<td>The organization feels that the team provides more feedback regarding software development issues.</td>
<td>The introduction of the agile process meant that the team more often informed the organization about software development issues, according to the organization.</td>
</tr>
<tr>
<td>The organization feels that the team has reduced the elapsed time before providing it with answers.</td>
<td>The introduction of the agile process meant that the team more quickly provided its surrounding organization with answers, according to the organization.</td>
</tr>
</tbody>
</table>

As shown in Table 11.2, the subjective analysis concluded that the organization perceived the introduction of the agile process positively. The team was considered more service-minded as it more often informed its surrounding organization with software development issues and answered more quickly on questions generated from the organization.

These positive aspects may be a result of the introduced changes in how the team developed software. As the team started to plan its work in detail it may have gained a better insight in how its work progressed and could therefore more easily inform the organization on software development issues. Further, the focus on the customer role may have made the team more service-minded and as the team started to discuss software development issues in pairs, knowledge was spread which made the team as a whole more competent.
11.5 Main Conclusions

Below, the main conclusions from the study are listed. They are subjectively derived and concern how agile development affects the collaboration between a software development team and its surrounding organization seen from an organizational point of view. The main conclusions are based on conclusions derived from interviews with customer representatives before, during and after the introduction of the agile process.

1. The introduction of the agile process did not affect the way or how often the customer representatives communicated with the software development team.

2. After introducing the agile process the customer representatives perceived that the software development team responded to them with answers more quickly.

3. After introducing the agile process, the customer representatives increased their trust to the software development team.

4. After introducing the agile process the customer representatives perceived that anyone in the team could more likely provide them with requested information when asked for.

The empirical data presented in this study is based on subjective opinions of those interviewed. However, to better understand the effects of introducing the agile process the quantitative data presented in Study 4.1 could be considered. An analysis of the data shows an improvement of productivity during the introduction of the agile process. Hence, we can argue that the introduction of the agile process had a positive effect when referring to the analysis of the quantitative data. This conclusion supports the views of the customer representatives as they argued that the team responded more quickly on questions and started to deliver software releases at agreed dates, when the agile process was introduced. Further, the customer representatives perceived that the team improved its level of knowledge which surely affects the team’s productivity in a positive way.

The effects from introducing the agile process presented in Study 4.2 are based on the team’s viewpoint. As stated by Murro et al. [67] it is important to recognize different perspectives when introducing software development processes in organizations. The team’s opinion was that the introduction of the agile process did not alter the way or how the team communicated with its surrounding organization. The team discussed software development issues with its customer representatives. The team perceived that the practice Planning game improved its understanding how it should plan its work. However, contrary to what is advocated by the standard XP, the customer representatives did not participate in the planning game since they
were busy and occupied with other work tasks. As stated by Beck [10], the planning game is an important tool for communication between customer and software engineers. Thus, it is possible that the collaboration between the team and its surrounding organization would have been even better if the customer representatives would have participated in the planning game.

11.6 Concluding Remarks

The results show that the introduction of the agile process had a positive effect on the collaboration between the software development team and its surrounding organization. A stated in the main conclusions, the organization did not perceive that it changed the way or how often it communicated with the team. Thus, the improved collaboration is a reaction of how the team changed its software development process. Earlier research conducted by Blomqvist et al. [12] has identified trust as a critical success factor in collaboration issues, which this research also supports. The organization increased its trust to the team as the team started deliver software at agreed dates and more frequently informed the organization regarding software development issues.

However, it is important to try to understand the underlying causes of why use of the agile process improved the collaboration between the team and its surrounding organization. A better understanding of this issue may help organizations realize how to introduce and adapt agile methods in order to maximize results from collaboration efforts. First of all, the fact that the team members were placed closer to each other probably played an important part, since this facilitated the team’s communication. Another factor was the introduction of the pair programming concept, where pairs helped themselves solve software issues. In our opinion, this contributed to an increased transfer of knowledge within the team. The planning game resulted in better time estimations from the engineers which may have contributed to the increased level of trust to the team from the organization. These activities and many more resulted in an improved performance by the team which was recognized by the organization, as it received the benefits of how the team developed software.

The results indicate that this informal approach to communication between the customer representatives and the team facilitated use of the agile process. Müller [64] characterizes informal communication as casual, spontaneous and can for instance appear in the form of ad-hoc conversations or memos. The credibility of the contents may be low, but the speed when communicating is high. Formal communication on the other hand is characterized as deliberate and impersonal with a high credibility of contents, but with a low speed when communicating. The style is in form of reports, briefings, tables or diagrams and so forth. Further, formal communication
can easily be reviewed and referenced to since its contents is under configuration control. Future research may investigate if agile methods perform best in environments where informal communication channels are used, or if software development results can improve as a result of introducing agile methods in environments where formal communication channels is the norm.

The overall conclusion from Study 4.3 is that the organization perceived that use of the agile process had a positive effect on its collaboration with the software development team. Thus, as the team adapted its software development process by introducing contents from XP, the overall software development performance in the organization improved. Although the result in itself is important (i.e. the collaboration improved as a result of using the agile method), the fact that the whole organization is affected when introducing a process is also important. The more interactive a team is with its surrounding organization, the more concern should be taken to not only fulfill the team’s needs but also the needs of the rest of the organization. Although there is no ambition to generalize the results from this study, they indicate that agile development facilitates collaboration between software development teams and their surrounding organizations. Hopefully, these results may inspire others to conduct research to further explore how agile development affects collaboration in organizations.
Chapter 12

STUDY 5: A SURVEY ON AGILE DEVELOPMENT

12.1 Background and Problem

In software engineering different processes, techniques and methods are used in different environments. There is a need to understand which processes etc. that are used in different situations, to what extent different processes are used, why they are used, how they have been introduced and, of course, how successful they are. This type of information can be obtained by carrying out surveys, as presented for example by Neill and Laplante [69], where the focus is on requirements engineering processes. Even if there are a few presented surveys in software engineering, there is still a large need for further surveys on different areas that are not yet enough investigated.

Extreme programming, XP [10], belongs to a group of software development processes called agile methods. These processes have gained much popularity in the software developing community during recent years. This popularity may depend on their ability to manage changes, which is important in software development, but also due to their incremental-based software development approach, which allows continuous review of a product while it is being developed. Though XP is popular, empirical investigations are still needed in order to clarify its impact in various contexts and environments, as mentioned by e.g. Layman et al. [59]. Thus, there is both a need for further case studies and for surveys.

It is not only important to understand what kind of results that may be expected from the usage of a software development process. It is also important to realize how a software development process should be introduced and in what environments it is suitable to apply. These aspects and more may be addressed if empirical data is collected and analyzed. The way a software development process is introduced to an organization affects its usage on a long-term basis. For instance, it is important that cultural and
organizational aspects are considered in order for a process to be accepted by an organization, as mentioned by Rea and Lientz [60]. Another important and related aspect is the adaptation of a process to fit the development environment of an organization. It is uncommon that a process is used "off the shelf", i.e. used without any modifications of the process as confirmed for instance by Beck [10] and Humphrey [41]. Usually the process is tailored to address software development needs and some parts of the process may not even be used. This also emphasizes the need to understand how different types of processes can be introduced as the introduction phase addresses adaptation issues.

In this study a survey is presented where 23 organizations in Sweden and Norway report their experiences from introducing agile development methods based on XP. The survey questions were based on typical issues in software process improvement and the results that were found in the case study presented in Study 4.1 to Study 4.3.

The organization of the study is as follows. Section 12.2 contains related work and Section 12.3 presents the research methodology. In Section 12.4, the characteristics of the survey participants are described. Section 12.5 presents analysis and interpretation of the survey results. Section 12.6 includes a summary of the survey results and concluding remarks.

12.2 Related Works

As software engineering is a relatively new engineering discipline, there is a lack of knowledge regarding the use of software development processes. This does not only concern results of applying certain practices in various contexts, but also to what extent different software development approaches are used. The need to gain deeper knowledge does of course not only apply to the area of software engineering but concerns other research areas in our society and surveys are useful for this purpose. This view is supported by Biemer and Lyberg [11] who conclude that the market for survey work has been growing the last two decades as a result of the expansion and evolution in many sectors in our society.

Some surveys have been performed in the area of software engineering, with the aim to investigate the use of software development processes and practices. Neill and Laplante [69] report on an exploratory survey where 194 individuals involved in software development in various ways participated in a web-based survey in 2002. The aim of the survey was to investigate the state of the art regarding use of software development processes. One interesting result from the survey was that the waterfall model was rather popular as 35% of the respondents answered that they used it. Another conclusion from the survey was that short projects tended to apply the waterfall model approach while longer projects reaching over two years in time more often
applied an incremental software development approach. However, the underlying reason for this is not discussed in the survey as the authors state that they merely present the results and have no ambition to draw any conclusions from the survey results. Another attempt to gain more knowledge on how software development is conducted was made by Cusumano et al. [22]. The survey conducted by Neill and Laplante sampled the respondents from a local region in America. The survey by Crandall et al. addressed software developers from all over the world as the aim was to investigate international differences in the adaptation of different development practices. The survey was web-based and the sample data comprised of 104 projects from India, Japan, America and Europe. The results show a major difference between the countries’ use of detailed designs, where India always used detailed designs compared to America where only 32% of the survey participants reported use of detailed designs. Another interesting fact was that only about a third on average from all countries applied the concept of daily builds, which means that each day a new version of a system is released, which offers opportunities for feedback and performance evaluation. This is rather surprising since the software development community in recent years has been stressing the need to develop software incrementally in order to respond to changes. The survey also covered the correlation between practice and performance. The data showed that having more complete designs appeared to produce fewer defects. This advocates use of more formal models such as the waterfall model as it uses complete specifications with defined milestones in projects and so forth. However, as the authors argue iterative development is better at managing change which has become more and more important in the global software development community. Thus, both approaches are useful seen from different perspectives. Like the analysis for the survey reported by Neill and Laplante, the analysis made by Crandall et al. was based on descriptive statistics. Thus, no statistical tests were used which makes hard to talk about significant differences. However, the results provide important information on the use and adaptation of software development practices.

During recent years, the research on agile methods has increased. There are numerous articles on experience reports on use of agile methods, for instance by Deias [23], Pine [75], Poole et al. [76] and Rasmusson [79]. They provide information on experiences from applying agile methods, which may be useful to consider when introducing agile methods to organizations. However, there is a lack of empirical studies on agile development which may improve our understanding of when and how to apply agile development in various contexts. This view is supported by Abrahamsson and Koskela [5] who state that the lack of empirical data hinders the ability to apply agile methods and modify them for different settings and domains. Rumpe [86] reports on a quantitative survey on projects where XP has been applied. The purpose of the survey was to get a general understanding of the cur-
rent situation in XP projects. The population target of the survey was all types of organizations around the world. The results are based on evaluating 45 questionnaires. Almost 90% of the respondents came from Europe and America. Further, over 80% of the respondents had used XP in teams of 10 people or fewer. The survey investigated difficulties with applying XP. The main part of the answers pointed out that the practices Metaphor and On-site customer were the most difficult to apply. The practice Metaphor was not used since many did not understand how to apply it. The practice On-site customer was not used since real customers were not available to participate with the software development team, as XP advocates. The survey also investigated which parts of XP that contributed most to success in a project. Here, three aspects were identified. The success factors were continuous testing, pair programming and XP’s focus on fulfilling the customer’s needs. When asked to identify risks with using XP, the main part of the respondents mentioned problems with acquiring a real on-site customer, opposition against XP in the organization and finally technology-related problems. These aspects identify similar problems derived from the case study which this survey addresses, presented in Study 4.1 to Study 4.3. For instance, the case study reports a lack of support from management which made it difficult for the team to implement the practice Continuous testing. Further, there was no real customer involved which probably reduced the effects from the collaboration between team and customer.

To summarize, there are not many surveys conducted on results and experiences of using XP in different contexts and environments. Especially, according to our knowledge there are no surveys conducted on how to introduce and adapt XP in various contexts although this is an important issue. Further, few reports or surveys regarding use of XP are based on scientific analyzes of collected empirical data, which may strengthen the validity of the results. The results from the survey presented in this study is based on both the use of descriptive statistics but also on the use of statistical tests. When reviewing the results from the surveys conducted by Cusumano et al. [22] and Neill and Laplante [69], it became obvious that although agile methods have gained in popularity they do not seem to be used as much as expected. Both surveys report that organizations to a high degree use more formal models such as the waterfall model, instead of agile methods based on iterative approaches. Another interesting aspect reported by Neill and Laplante is that projects with a duration period of over two years most often apply an incremental development life cycle. This is twice as common as it is in shorter projects. A possible explanation for this is that it is easier to provide a good overview of what shall be done in shorter projects, thus making it possible to state project milestones and so forth which occur in more formal models such as the waterfall model. However, in longer projects this is not as simple as it is difficult to state what shall be delivered and when. This situation suits an incremental life cycle as software is delivered
in regular intervals as new requirements emerge. The reported low usage of software development approaches based on agile development ideas may depend on the fact that little is known regarding using agile methods in various contexts. This issue is addressed in this survey, as results on introducing and adapting agile methods in different environments are reported.

12.3 Research Methodology

12.3.1 The Case Study

The purpose of this survey is to investigate Scandinavian software development organizations’ experiences of introducing and adapting agile methods. The research question is formulated as:

What are the results of introducing agile methods based on XP into software development organizations?

However, the results of the survey are relevant for other software development organizations to consider, maybe especially for countries located in western Europe where society and culture in many ways may be similar to Sweden and Norway. In these cases, it is advised to review the results of the survey to investigate how they fit in the specific context of the current country. The survey questions were based on typical issues in software process improvement and the results from performing a case study where a process based on XP was introduced to a large software development organization, described in Study 4.1 to 4.3. The case study generated a number of conclusions concerning this issue, presented in Section 12.5.3. The survey investigates the amount of support these conclusions receive from different software development organizations in Scandinavia. In the rest of this section, the purpose and background of the case study is provided.

The purpose of the case study was to gain knowledge how the process was adapted and used in an environment that differs from the ideal context for using XP, as argued and described by Beck [10]. This ideal context consists of new development of code preferably not based on legacy systems, which is led by one deeply involved customer who is with the team all the time and has good insight of the end product’s characteristics. However, our experience is that this situation seldom exists in modern software development organizations. This was the situation with the case study as the organization maintained and supported many systems and had several customers all over the world. Further, there existed a number of internal projects and new versions of systems were released many times a year. Thus, the case study provided insight how an agile method was introduced to a complex but in our opinion rather common software development environment. The results from the case study are important, as there is a lack of empirical data on
usage of agile methods in different contexts. By observing and measuring
effects from using agile methods we may gain a better understanding how
and when to use these methods. In the case study, a software development
team used a process based on XP during eight months. One practice at a
time was introduced to the team. After each completed iteration, which was
a time period of two weeks, was a new practice introduced.

The case study consisted of three studies. One of the studies concerned
quantitative measurement of the performance of the team that was intro-
duced to the process. One study addressed how the team perceived the
adaptation and usage of the process where qualitative data was collected in
form of interviews and observations of the team. The third study investi-
gated how the team collaborated with its surrounding organization as it used
the agile method, seen from an organizational perspective. It is important
for a software development organization as a whole, to deliver software with
high quality. Thus, it is important to address internal collaboration issues
in an organization for it to improve its performance. This was addressed in
the study as people in the organization which collaborated with the team,
were interviewed to investigate how they perceived that their collaboration
with the team changed as the team started using the agile method. The
collected data, which came from different perspectives in the organization,
helped realize how processes based on XP may be successfully introduced
to similar organizations.

12.3.2 Survey Method

The target population for the survey was software development organizations
from Norway and Sweden. These countries have similar cultures, beliefs and
values in their societies as described by Kangas and Palme [52], and this is
also true for the software development community according to our expe-
rience. The similarities in the cultures minimize any cultural aspects that
may bias the results. Thus, the target population is various software devel-
onegment organizations in Scandinavia. The survey participants are described
in more detail in Section 12.4. The purpose of the survey is twofold. One
objective is to investigate organizations’ experiences from introducing pro-
cesses based on XP. As a consequence, all different types of organizations
were included in the survey’s target population. These organizations are of
different sizes and complexity, which had introduced processes based on XP
in some way. The term based on XP is used since it is common that XP is
adapted in some way or that parts of the process is not used, as mentioned
in Section 12.1. The second objective is to investigate to what extent other
organizations support the findings from the case study. The findings concern
how to introduce and adapt processes based on XP in organizations, and
input from other organizations whether they share these findings is impor-
tant when trying to understand cause and effect from introducing this kind
of process in different environments. The number of organizations that have introduced and used processes based on XP are rather few according to our experience. Organizations of interest were identified through contacts in the agile community or through agile news groups such as [3]. 23 organizations participated. It is difficult to estimate how many organizations that have introduced processes based on XP. Thus, it is uncertain if the sample data adequately represents the target population. A more detailed discussion regarding if and how the sample represents the target population is provided in Section 12.4.1.

The survey included questions concerning issues of introducing and adapting agile methods in various contexts. A summary of the questions whose answers were used in the analysis is provided in Table 12.1. The questions addressed four different areas. Part A contains questions that addressed the background of the survey participants which helped understand the survey results better. Part B contains questions that addressed the introduction and use of agile methods based on XP. This area dealt with issues such as what was difficult versus easy in XP to introduce and why. Then, Part C includes questions that addressed results of using XP. Finally, Part D contains questions addressing teams’ experiences of using XP and whether they will use it in the future and so forth. The questions were designed to address important issues when introducing processes based on XP into organizations. The questions were answered with free text or by choosing an alternative on a three or five graded scale. The scale was based on the Likert scale type described by Barnett [8]. It is an ordinal rating scale measuring the strength of agreement with a clear statement. Respondents are asked to indicate the amount of agreement or disagreement, for instance from strongly agree to strongly disagree.

As Biemer and Lyberg [11] mention, an important step for achieving optimal quality in a survey is to perform a pilot survey to obtain information that can improve the main survey. Therefore, a pilot survey was conducted on the team that participated in the case study that this survey addresses. After analyzing the results from the pilot survey, some changes were made to the main survey. The survey was conducted over the internet, as an electronic mail survey described by Biemer and Lyberg [11]. This form was chosen mainly due to economical and time constraints. However, an important positive aspect with an electronic mail survey is that interviewer variance is eliminated, as mentioned by Biemer and Lyberg [11]. Interviewer variance is the variability between the systematic biases of interviewers, that is how interviewers may influence responses to questions. Further, the electronic mail survey form was favored since the survey participants were dispersed over a wide geographical area. The other option, to perform a survey face to face with the respondent, may mean that the respondent detects or imagines nuances in how the interviewer asks the questions which in turn may affect how the respondent answers. This risk is eliminated with the use of
Table 12.1: Summary of survey questions.

<table>
<thead>
<tr>
<th>Part A: Background of study participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What type of projects was XP applied on?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part B: Introduction and use of XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Did the team perform any investigation whether parts of XP were already used, before it was introduced?</td>
</tr>
<tr>
<td>2. State the practices that were in use before XP was introduced. Further, state the practices that were introduced and adapted?</td>
</tr>
<tr>
<td>3. State which practices were easy versus difficult to introduce?</td>
</tr>
<tr>
<td>4. How did the team’s type of development affect the introduction of the practice Collective code ownership?</td>
</tr>
<tr>
<td>5. How did the type of organization which the team was active in affect the introduction of the practice Continuous testing?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part C: Effects from using XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What effects do you think the use of XP contributed with?</td>
</tr>
<tr>
<td>2. To what extent were the people the team collaborated with aware that the team was using XP?</td>
</tr>
<tr>
<td>3. Do you think the team changed its way of communicating with the surrounding organization as a result of using XP?</td>
</tr>
<tr>
<td>4. Do you think the surrounding organization’s attitude towards the team changed as a result of the team using XP?</td>
</tr>
<tr>
<td>5. Were there any problems associated with the use of XP’s practices?</td>
</tr>
<tr>
<td>6. Do you think incoming support issues during an iteration should be managed by the practice Planning game?</td>
</tr>
<tr>
<td>7. How did the team’s type of development affect the attitude to the practice Planning game?</td>
</tr>
<tr>
<td>8. Do you think the team’s type of development contributed to a different way of using the practice Simple design?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part D: Experiences from using XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Which, if any, practices were introduced and used all the time?</td>
</tr>
</tbody>
</table>
an electronic mail survey. The survey participants were reminded twice to complete the survey, the first time after three weeks and the final time after six weeks. Further, they were contacted if there was a need to clarify any contradictions in the survey data, or if some data was missing. The response frequency was about 85%, as 23 of the 27 survey participants completed the survey. This is a high response rate, as a common value is about 50 to 55% in surveys of similar character according to Biemer and Lyberg [11]. The high response rate indicates that an electronic mail survey was appropriate to use in this study, maybe as it were easy for the respondents to answer the survey and return it. The high response frequency increases the reliability of the survey results as the main part of the respondents participated in the survey.

The purpose of the survey is to investigate Scandinavian organizations’ overall experiences of introducing processes based on XP. This issue is addressed as the survey investigates the experience of the survey participants regarding introducing and adapting agile methods. Further, the survey investigates the amount of support the conclusions from the case study, presented in Study 4, received from the organizations. The results of the organizations’ overall experience of introducing and adapting agile methods are presented in Section 12.5.2. The main result from this analysis was that the ideal context in which to apply XP, as described by Beck [10] is not common in the software development community. The results show that not all of XP is introduced and only two thirds of XP’s practices were applied continuously. A number of observations were identified by comparing survey results with the use of descriptive statistics. To increase the validity of these observations, results from other case studies and surveys were included in the discussion. Descriptive statistics were used since the aim was to identify issues which a majority of the survey participants supported. These observations are not associated with the former case study. They emerged while analyzing the survey data.

When analyzing the survey data, non-parametric tests were used since the data was at most at an ordinal scale which does not allow parametric assumptions of the underlying population. The statistical tests Chi Square method and the Fisher’s Exact test, for instance described by Motulsky [66], were used when comparing observed distributions of independent data on a nominal scale to so called expected frequency distributions. These tests were useful when investigating whether an answering alternative received more responses than expected, when addressing a conclusion consisting of two survey questions. In addition to that, descriptive statistics were used when appropriate, for instance to identify which answering alternative in a question that received a majority of the answers. Factor analysis and the Spearman method, for instance described by Motulsky [66], were used as additional approaches to find relationships between the survey questions. Factor analysis concerns the identification of data relationships, while re-
ducing the amount of data. This approach is useful as a lot of data may be
difficult to overview, and by applying factor analysis data can be reduced
while connections between data sets can still be identified. The Spearman
method summarizes the strength and direction of a relationship between
two variables. However, these analyzes did not generate any significant re-
lationships. The conclusions drawn from the case study are presented in
Section 12.5.3. After each conclusion follows an analysis of the survey data
of interest to investigate whether the organizations support the conclusion
or not.

12.4 Participant Characteristics

This sections contains a background on the survey participants. This infor-
mation is useful when analyzing the results of the survey, as knowledge is
gained on the characteristics of the people and organizations that made up
the sample data. The survey participants covered a broad spectrum of organ-
izations and positions in software development. Below, a data analysis on
the survey participants’ responses regarding their background is provided.
The average time spent developing software with XP was 20 months, where
the minimum time spent with XP was 1 month and the longest was 8 years.
However, this long time period of 8 years probably refers to software devel-
opment based on similar ideas included in XP as the method was published
6 years ago. The experience of the typical software engineer that used XP
is presented in Figure 12.1.

![Experience of the typical software engineer.](image)

As shown in Figure 12.1, the typical software developer was rather ex-
12.4. PARTICIPANT CHARACTERISTICS

experienced in software development. The term experience refers to general experience in software development and not specifically concerning various agile development issues. The majority of the respondents graded the typical software engineer in their teams to have between 3 and 5 years of software development experience. Thus, the conclusions from the survey are based on experienced people who were introduced to XP.

Figure 12.2 presents the number, and different kinds, of development contexts where XP was applied in. As shown, most of the survey participants focused on developing new functionality, while less than a forth maintained and supported legacy systems. In some cases both new development and maintenance were performed by the survey participants.

![Figure 12.2: Development contexts where XP was applied in.](image)

The marks in the category Other refers to adaptations of existing systems to new technical platforms or software evolution based on existing code kernels. A mark means that a respondent has chosen a certain alternative for a question in the survey. As seen in both Figure 12.1 and Figure 12.2, the number of marks exceed 23 which is the number of the survey participants. This is caused by the fact that some respondents answered with more than one category.

Figure 12.3 presents the opinions the teams had regarding XP before the agile process was introduced. This information is useful as their attitudes towards XP before it was introduced were compared to their attitudes after it had been introduced. Further, it was important to know their attitude towards XP before it was introduced since that probably affected their answers in the survey to some extent. The number of marks in Table 12.2 and Table 12.3 sometimes is not equal to 23. This depends on the fact that either did not all of the participants answer some question, or that some of
the participants may have answered with two marks on a question.

![Bar chart](image)

Figure 12.3: The teams’ opinions of XP before it was introduced.

The majority of the respondents were positive to XP before it was introduced, although a large number was neutral to XP. It is interesting to notice that none was negative to XP before it was introduced. This may be a result of the intense positive focus that agile methods have received during the last years. However, it can also be a result of the fact that it were the teams themselves who wanted to introduce XP thus less probability that they would be skeptical about XP. Further, another explanation may be that only one person from the teams participated in the survey, thus any negative concerns regarding XP shared by other team members may not have been expressed in the survey. Figure 12.4 presents the initiators for introducing XP. As shown, the dominant cause for introducing XP was an interest from the teams themselves.

As it were the teams themselves that most often took the initiative to introduce XP a positive enthusiasm regarding agile development may be expected, which is also confirmed in Figure 12.3. To understand why the teams wanted to try XP, more information is needed. This information is provided in Table 12.2, where the teams’ reasons for introducing XP are presented. No affect means that the reason in question did not motivate the team to introduce XP, while large affect strongly motivated the team to introduce XP. After reviewing the data in Table 12.2 it can be seen that important reasons for introducing XP are to improve the management of changes in software development, and using a process which delivers visible results.

Table 12.3 contains the respondents’ answers of how important different types of quality characteristics were for the products developed by use of
12.4. PARTICIPANT CHARACTERISTICS

Figure 12.4: Initiator for agile development.

Table 12.2: Reasons for introducing XP.

<table>
<thead>
<tr>
<th>Reason</th>
<th>No affect</th>
<th>Little affect</th>
<th>Large affect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interested in XP.</td>
<td>5</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Wanting control over process.</td>
<td>1</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>Wanting reduced cycle time.</td>
<td>5</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Wanting better management of change.</td>
<td>1</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>Wanting a more adaptable process.</td>
<td>4</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>Wanting a more understandable process.</td>
<td>5</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>Wanting more visible results.</td>
<td>2</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>Wanting a process that is accepted by the team.</td>
<td>4</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

XP. The sign -- means that the quality characteristic in question is very unimportant, and the sign - - means that the quality characteristic in question is unimportant. The same line of reasoning applies on the signs + and + +. These quality attributes are often referred to when measuring quality in software development products, as mentioned by Sommerville [92]. As seen in the table, dependability and maintainability are important characteristics for most of the respondents. This supports the choice of XP since it is argued for instance by Beck [10] that XP helps maintain software and handle defects by applying test-driven incremental software development. In Table 12.2 and Table 12.3, the number of marks for an alternative does not always equal
the number of study participants (i.e. 23). This depends on the fact that some participants either did not answer or answered with two marks on an alternative.

Table 12.3: Quality characteristics for developed products.

<table>
<thead>
<tr>
<th>Quality characteristic</th>
<th>- -</th>
<th>-</th>
<th>No affect</th>
<th>+</th>
<th>+ +</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependability</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Usability</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Efficiency</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Maintainability</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Portability</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Reusability</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

12.4.1 Validity

Sources of error when conducting a survey can be divided into non-sampling errors and sampling errors, as stated by Biemer and Lyberg [11]. Non-sampling errors concern errors that do not relate to the size of the sample but to the sample itself. For example, people that participate in a survey may not tell the truth if the question is politically sensitive and so forth. This type of error was addressed in the design of the survey by formulating non-leading questions of a neutral character. Sampling errors relate to how sample data is chosen and how much of the population that is covered by the sample data. For example, a sampling error may occur if the sample size is too small compared to the target population, which decreases the validity of the results as the sample data might present a false picture of the target population’s opinion. Three aspects affect the reliability of results from a survey. These are the population size, the type of sampling done and the sample size. The population size of this survey is unknown, but an adequate estimate may be that there exist at most 50 to 100 companies in Norway and Sweden who has introduced a process based on XP. Through contacts in the agile community and through agile news groups 27 companies that had introduced a process based on XP were identified. The other aspect (i.e. the type of sampling done) should preferably be based on random sampling where each sample data is equally likely to be included in the survey. However, based on the small number of sample data this option was not applied. Instead, all of the companies were asked to participate in the survey. The third aspect that affects the reliability of survey results concerns the sample size. Based on the assumption that the target population consists of 50 to 100 companies, then the sample size covers roughly 30 to 50% of the population which is a high number. Thus, although the sample data included in the survey was not randomly chosen the reliability of the results
may still be considered high since a large part of the target population is believed to be covered by the sample data.

To increase the validity of the results, a pilot survey was conducted before running the real survey. The purpose of the pilot study was to identify potential problems in the survey design. The survey was pretested by using a so called think-aloud interview and an expert review, which are two proposed techniques mentioned by Biemer and Lyberg [11] to reduce the survey error. Both techniques help identify questions that may be misunderstood, complicated instructions or other questionnaire problems. Think-aloud interview means that the respondent says what he/she thinks so that the interviewer can recognize problems in the survey such as comprehension and recall errors. A think-aloud interview was conducted with one person from the team in the case study to gain information of potential survey design problems. Expert review refers to letting a person which possesses substantial knowledge in some area of interest, review a survey in order to find survey inconsistencies. In this case, a colleague who had knowledge about XP performed the expert review in order to provide information regarding the survey design from an agile process point of view.

12.5 Results

12.5.1 Analyzing Survey and Case Study Results

In the survey analysis, presented in Section 12.5.2, the results from the survey is compared with results from the case study and results from research conducted by others. This approach makes it possible to review the survey results on a broader basis, as the survey results are compared to results generated in different contexts and by other researchers.

The analysis presented in Section 12.5.3 investigates to which extent the data gained from the survey supports the conclusions generated from the case study reported in Study 4. Thus, the purpose is to address the validity of the case study results, as the conclusions form the case study are proved on a larger population in form of the organizations that participated in the survey.

12.5.2 Survey Analysis

The aim of the survey is to gather more information regarding how processes based on XP are introduced into organizations and what the results of these introductions are. This was achieved by analyzing and comparing results from the survey with results from the case study and related research. Both the survey and the case study provide unique results that add to the body of knowledge in this area, and by comparing these results an improved understanding how to introduce processes based on XP may be
gained. After analyzing the survey and the case study results with the use of descriptive statistics, a number of observations emerged. Further, results from other surveys and case studies performed in this area were also considered when formulating the observations, in order to increase the validity of the observations.

Descriptive statistics were used when analyzing the survey data. The motivation for this was that by detecting trends in the data which a majority of the respondents supported, conclusions could be identified. This unbiased approach helped detect conclusions that may have been missed if the research had followed a more detailed design. Statistical tests were used when considering the validity of the findings from the case study. The motivation for this was that it was desirable to determine whether the findings were statistically significant when compared to the data collected in the survey. By combining these two groups of results more reliable results are produced as the results are based on both data from a larger population (i.e. the survey) and data from a smaller population (i.e. the case study). All of the observations presented below have in common that they are based on results that a majority (i.e. 58% or more) of the survey participants supported.

1. In the survey a majority of the introduced practices were not adapted. In the case study, a majority were adapted. A possible explanation for this may be that a majority of the survey participants developed new code, whereas the team in the case study mainly maintained and supported code. Another explanation may be that the development environment in the case study was more complex than the ones in the survey, where a complex environment with many systems and customers may contribute to a higher level of adaptation. A survey on the adaptation of development methodologies conducted by Fitzgerald [32], reports that 58% of the 162 respondents that participated in the survey did not follow a methodology rigorously, that is using a process without any adaptations. The reasons for not following a method rigorously were that methodologies may be cumbersome to apply in the sense that they consume time and resources which may not be available. Further, how a method is used depends on the needs of the actual situation, that is developers tend to use their own judgement how to apply a process depending on the actual development environment at hand. When considering the results reported by Fitzgerald agile methods are then more likely to be adapted as a result of the actual contexts where they are applied in, instead of lack of time or resources, as they generally are considered rather easy to introduce.

2. A majority of the survey respondents stated that XP’s practices were introduced a couple at a time or one at a time instead of all at once.
This observation was also true in the case study. A possible explanation for this may be that it is difficult for a team to embrace all of XP at once, when the team is occupied with its regular work. Further, since this observation is true for many different types of environments is it possible that the observation is valid regardless how the development environment is defined. Other case studies that address introduction issues regarding processes based on XP, such as the ones reported by Grenning [38] and Rasmusson [79] do not describe whether parts or all of XP were introduced at once. This is unfortunate since the way XP is introduced may have an impact on if and how XP is used, as reported in Study 4.1 to 4.3 and confirmed by the results from this survey. However, Schuh [88] reports on a study where XP was introduced to an organization where XP’s practices were introduced a couple at a time. Schuh states that the reason for this was the lack of available time to learn and adapt the practices to the environment at hand. This statement supports the results from the case study described in this survey, where XP’s practices were introduced one at a time since the team did not have time to spend on learning and adaptation issues.

3. A majority of the survey respondents stated that they would continue to use XP in the future. In the case study, a majority stated that they would not continue to use XP in the future, with the exception of pair programming which was considered useful when transferring knowledge from one person to another. Further, the practice Collective code ownership was considered useful since it increased the team’s level of competence. A possible explanation for this difference of opinion may be that in the survey a majority stated that it was the teams themselves that wanted to try XP whereas in the case study the decision was made by management, although the team was interested in agile development. Thus, personal motivation seems to be important if a process is going to be applied continuously. It should be mentioned that whether some practices are used or not is not necessarily based on the willingness of the software development team using the process. For instance, the practice On-site customer is often implemented with the help of a proxy customer, that is people in an organization act the role of the customer which was the situation in the case study presented in this survey, and in other studies such as the one presented by Cao et al. [16]. In these cases, it is the resources and willingness from the surrounding organization that decide whether the practice will be used. Further, as mentioned by Cao et al. [16] some practices, such as pair programming, are more than others tied to the personalities of the software engineers. Thus, in the case of the practice Pair programming some may prefer to sit next to each other and develop
software since they excel in a more social environment, while other will not pair program due to different reasons (e.g. lack of social skills).

4. There is no consensus between the results from the survey and the results from the case study regarding which practices were easiest to introduce. This is also true when only comparing results from the survey. This observation supports the line of reasoning made in Section 12.5.3 where it is concluded that it is difficult to state how a process will be introduced and what the effect from that introduction will be. A logical approach mentioned by Schuh [88] is to first introduce the practices addressing the most urgent needs. In these cases, it may be more important to phase in a practice early even if it is considered difficult to introduce since it will, once implemented, help the team develop software in a better way.

5. A majority of the survey respondents stated that it is difficult to implement the practice On-site customer with a real customer. This observation was also true in the case study. The survey results identified the practice to be the most difficult to introduce, whereas the results from the case study ranked it as the fifth most difficult practice to introduce. This difference of opinion may be based on the fact that in the case study the customer did not participate as much as advocated by XP. Thus, the practice may be easier to introduce as it did not require much effort to implement. The solution to implement the practice On-site customer with people from the organization acting the customer role instead of a real customer, is common also in other studies such as the ones conducted by Grenning [38], Schuh [88], Rasmusson [79] and Cao et al. [16]. It is interesting to notice that so few studies use real customers when implementing the practice On-site customer. It may be that the practice is not applicable in most software development contexts since many companies have several customers often located in other countries and so forth, which obviously complicates an implementation of the practice as advocated by XP.

6. The practice Metaphor was regarded as difficult to introduce when reviewing the results from both the survey and the case study. A majority of the survey participants did not use the practice Metaphor. The practice was also not used in the case study. Problems of implementing the practice Metaphor are also reported by Grenning [38], Schuh [88] and Rasmusson [79]. The underlying reason for this seems to be the difficulty in developing a meaningful metaphor, that adequately describes the system or systems being developed. It may be
difficult to understand how to use the practice and what the concept of a metaphor means, especially in a software development context. Further, in many cases an organization already has an established software architecture that is used when designing the evolution of a system, which eliminates the need of a metaphor.

7. A majority of the survey participants did at least already use a third of the practices before XP was introduced, in the sense that they performed activities that were similar to the practices in XP. In the case study, four practices were already used in some way or the other before XP was introduced. Thus, it seems that a large part of XP’s contents may already be in use in many organizations. This observation is however not entirely supported in the survey regarding XP usage in software development projects, conducted by Rumpe et al. [86]. 45 respondents participated in the survey and shared their experiences of using XP in different kinds of projects. They allocated numbers to XP’s practices in correlation to how much they were used. 9 meant that the practice in question was strongly used and 0 meant that the practice was not used at all. Thus, the higher the number a practice received, the more it was used. Ten of the practices received average values between 6 and 8 and only two practices, On-site customer and Metaphor, received average values of 4 versus 3. The high values may indicate that these organizations did not prior to the introduction of XP perform any activities similar to XP’s practices since they stated that the practices were strongly used. However, it can also mean that the practices were introduced even if the organizations already performed similar activities as the practices may have complemented the already established activities in some way.

8. The results from this survey state that 71% of the practices introduced are applied continuously. This result indicate that mostly only parts of XP are applied at a continuous level in many organizations. However, in the case study all of the nine introduced practices were applied at a continuous level. This may depend on the fact that the team reviewed its development environment thoroughly before introducing XP, thus identifying specific needs that parts of XP could fulfill. There is a debate in the agile community how much of XP should be used, to still be able to call it XP. This issue is addressed by Sharp and Robinson [90] where they discuss the importance of each practice and how the practices correlate to XP’s values. However, this debate is not a high priority in this research as the focus is on what is actually introduced, and the results from using that process. It does not matter whether
9. The results from the survey suggest that there does not seem to be any pattern regarding the attitude towards XP and how much of XP that is used. The results from this survey show that there is no correlation between the number of practices used continuously, and the attitude towards XP. Teams that only applied three or four practices had both a positive and negative view on XP. This was also true for teams that applied most of the practices. The same line of reasoning is also valid for teams that applied roughly half of XP, for instance seven or eight of the practices. Thus, the joy of using XP is not likely associated with how much of XP that is used. It is more important to focus on how XP can satisfy software development needs, regardless the number of practices this will require. This view is supported by Grossman et al. [39] who report on experiences of introducing XP. Their ambition was to implement all of XP’s practices but although this did not happen, they report that the process seems to fulfill the identified needs. Further, it is interesting to notice that none of the participants in this survey used all of XP. This was also true for the case study. Thus, if the requirement for calling a process XP is that all practices in XP should be applied, then few people seem to be using XP in the software development community.

10. A majority of the survey participants wanted a process that can manage changes in a good way when the requirements have not been specified in advance, and a process that delivers visible results. As a majority of the survey participants were more positive to XP after than before it was introduced, it is likely that processes based on XP fulfill these requirements. The author of this thesis believes the practice Planning game is an important tool when managing changes in software development. Further, the close collaboration with the customer in conjunction with the use of diagrams to inform the team of its progress surely improves the visibility of delivered results. The survey results provided by Rumpe et al. [86] identify XP’s focus on testing as the main success factor when using XP. This view is also supported by the results from this survey and the case study. However, these results also show that the practice Continuous testing requires both time and effort to implement which should be considered when planning how to introduce processes based on XP.

It is interesting to notice, as stated in the observations, that XP’s practices are mostly introduced one by one or a couple at a time, and not all at once. Further, not all of the practices are used and roughly two thirds of
those introduced are applied continuously. Thus, the ideal situation where all of XP is introduced at once and all practices are applied continuously as described by Beck [10], does not seem to be very common. This could imply that some practices in XP are not that appreciated by the software development community. Another interpretation may be that some of XP’s practices are already in use in the organizations, as mentioned in observation number 7.

12.5.3 Validating the Case Study Results

In addition to investigating various organizations’ experiences of introducing and adapting agile methods, the aim of the survey was to investigate whether the organizations supported the conclusions from the case study, which this survey addresses. The conclusions are presented below. After each conclusion follows an analysis based on the survey results, in order to determine whether the conclusion is valid. As the survey investigated whether the organizations shared the findings from the case study, there is a relationship between the survey questions and the conclusions. The procedure for validating a certain conclusion is to analyze the data from its associated question or questions. For each conclusion, the questions used are identified and motivated, the applied analysis method is identified and the results presented. The questions are presented in Table 12.1. It should be noted that there may exist other questions in the survey that could be used when investigating if the organizations supported the conclusions. When comparing observed distributions of independent data on a nominal scale, either the Chi Square test or the Fisher’s Exact test was used. To apply the Chi Square test the data should be on a nominal scale and there should be enough of data points. If there is not an adequate number of data points which motivate use of the Chi Square method, then the Fisher’s Exact test is used instead as it only requires one data point in a row or column, which was fulfilled by the sample data. However, for both tests it is difficult to detect a statistically significant difference in the data sets due to the low number of data points. A more detailed presentation of the statistical tests is provided for instance by Motulsky [66]. For all tests, a significance level of 0.05 was used when deciding whether to reject or accept the null hypothesis.

1. Before a process based on XP is introduced to an organization, an evaluation of the organization’s development processes should be conducted.

The survey results show that XP’s practices are introduced most often a couple at a time, as roughly 50% of the respondents did this. The other options, either one practice at a time or all of the practices at once, were only supported by 25% each of the respondents. A motivated question to ask is whether the survey participants more often
performed process assessments prior to the introduction of the agile method, when parts or all of XP were introduced at once. When introducing parts or all of XP at once, there are higher demands on the teams to know how to adapt and apply the practices in order to avoid chaos, since there are many practices that need to be addressed simultaneously. This activity may be facilitated by performing evaluations of the teams’ processes before XP is introduced, as the teams will better understand their needs and how XP’s practices may be adapted in order to provide maximum support. This approach was used by the survey participants as no thorough assessment was done and only one general assessment performed when introducing one practice at a time. However, when introducing a couple of practices at a time or all of XP at once, there were three thorough assessments and eight general assessments conducted.

Survey results for questions B1 and B4 were analyzed. B1 was used since it addresses whether the teams performed any process evaluation prior to the agile method was introduced. B4 was used since it addresses the level of difficulty associated with the introduction of XP’s practices. The hypothesis is that a thorough evaluation will mean reduced difficulty when introducing the practices as the requirements and needs of the team are better understood. The Chi Square test was used. The calculated total chi square value was 2.7. However, since the related critical chi square value was 9.5 the null hypothesis was not rejected. Thus, there was not a statistically significant relationship between the variables. This means that the analysis based on these data, does not show that there is a connection between how easy the introduction of XP was and whether the teams performed any investigation of their development processes before XP was introduced.

2. The introduction of the practice Continuous testing is more difficult in maintenance and support environments.

Ten of the respondents worked in maintenance and support environments. Out of these, one answered that this type of environment had a negative impact on the introduction of the practice Continuous testing. However, seven answered that the environment facilitated the introduction of the practice. Further, 20 answered that they mainly worked in environments where new functionality was developed. Here, none answered that the environment had a negative impact on the introduction of the practice Continuous testing, and 15 answered that the environment instead had a positive impact on the introduction of the practice. Thus, as mentioned below the data does not support the conclusion that the introduction of the practice Continuous testing is
more difficult in maintenance and support environments.

Survey results for questions A1 and B6 were analyzed. A1 was used since it addresses which types of development the teams performed. B6 was used since it addresses if and how the organizations affected the introduction of the practice Continuous testing. The hypothesis is that it is more difficult to introduce continuous testing in a maintenance and support environment since large efforts are often invested in locating and correcting software faults, but with limited changes which may make it difficult to validate the introduced change. Further, as was the case in Study 4, such an environment often includes several systems and platforms which implies additional efforts to implement test frameworks for each system. The Fisher’s Exact test was used. The calculated p value was 1. It was not possible to find a significant difference between the organizations’ impact on the introduction of XP (described by B6) and the types of development processes based on XP were introduced into (described by A1). This means that the analysis based on these data does not show that there is a connection between how easy it is to introduce the practice Continuous testing, and whether the team performed new development of code or maintained and supported code.

3. A development environment working with support issues facilitates introduction of the practice Collective code ownership.

Ten of the respondents worked in maintenance and support environments. Out of these, two answered that this type of environment had no impact on the introduction of the practice Collective code ownership. However, eight answered that the environment facilitated the introduction of the practice. Further, 20 answered that they mainly worked in environments where new functionality was developed. Here, 7 answered that the environment had no impact on the introduction of the practice Collective code ownership, and 13 answered that the environment instead had a positive impact on the introduction of the practice. Thus, 80% of the answers from respondents that worked in maintenance and support environments were of the opinion that this type of environment facilitated the introduction of the practice Collective code ownership. This can be compared to 65% of the answers from respondents that worked in environments where focus was on development of new functionality, who were of the opinion that this type of environment facilitated introduction of the practice. However, the data does not show that the introduction of the practice Collective code ownership is easier in maintenance and support environments.

Survey results for questions A1 and B5 were analyzed. A1 was used
since it addresses which types of development the teams performed. B5 was used since it addresses if and how a team’s type of development affected the introduction of the practice Collective code ownership. The hypothesis is that it is easier to introduce collective code ownership in a maintenance and support environment, since such an environment invites the teams to search through the systems to identify the faults that caused the unwanted behavior. The Fisher’s Exact test was used. The calculated p value was 0.2. Thus, the null hypothesis was not rejected. This means that the analysis based on these data, does not show that there is a connection between how easy it is to introduce the practice Collective code ownership and whether the team performed new development of code or maintained and supported code.

4. A development environment working with support and maintenance issues complicates introduction of the practice Simple design.

Seven of the respondents answered that they worked in maintenance and support environments. Out of these, two answered that this type of environment complicated the introduction of the practice Simple design. The rest of the respondents that worked in maintenance and support environments stated that their type of environment did not affect how the practice was introduced. 18 of the respondents answered that they worked mainly with development of new functionality, and only two of these stated that their type of environment complicated the introduction of the practice Simple design. The rest of the respondents that worked mainly with development of new functionality answered that their type of environment did not affect the introduction of the practice. Thus, roughly a third of the respondents that worked in maintenance and support environments stated that their type of environment complicated the introduction of the practice, whereas only about 10% of the respondents that worked mainly with new development answered that their type of environment complicated the introduction of the practice Simple design. However, as calculated below the data does not show that the introduction of the practice Simple design is more difficult in maintenance and support environments.

Survey results for questions A1 and C8 were analyzed. A1 was used since it addresses which types of development the teams performed. C8 was used since it provides an answer to whether the teams thought that their type of development contributed to that the practice Simple design was applied differently than expected. The hypothesis is that a maintenance and support environment complicates the introduction of the practice. It becomes more difficult to use designs that are as simple as possible as the developers must think in advance for fu-
structure functionality and coupling issues between different systems. The Fisher’s Exact test was used. The calculated p value was 0.93. Thus, the null hypothesis was not rejected. This means that the analysis based on these data, does not show that there is a connection between how easy it is to introduce the practice Simple design, and whether the team performed new development of code or maintained and supported code.

5. It is difficult to introduce the practice On-site customer without adaptations.

Survey results for questions B3 and C5 were analyzed. B3 was used since it addresses if and how XP’s practices were introduced. C5 was used since it addresses problems when using XP’s practices. The hypothesis is that it is difficult to introduce the practice On-site customer without adaptations, as real customers seldom are available to participate with the teams. The results were analyzed by using descriptive statistics since the purpose was to investigate whether a majority of the survey respondents stated problems when introducing the practice On-site customer. Seven out of the 23 respondents did not introduce the practice On-site customer at all. Motivations for this were, according to their answers, that the real customer was not available (e.g. working in another country) or that people in the organization did not want to act the on-site customer role. Further, 11 of the respondents did introduce the practice but with adaptations, in the sense that people in the organization such as project managers took on the customer role. Further when applying these solutions, it was common that the person playing the customer role did not participate with team all the time as the practice advocates. This scenario with a proxy customer is also common in other research papers mentioned for instance by Murro et al. [67]. Only five of the survey respondents stated that they applied the practice as advocated by XP. Thus, a majority of the respondents stated difficulties with introducing the practice On-site customer without adaptations.

6. A development environment working with maintenance and support issues reduces meaning of the practice Planning game.

Ten of the respondents worked in maintenance and support environments. Out of these, five answered that this type of environment had no impact on their attitude towards the practice Planning game. Five answered that this type of environment had a positive impact regarding their attitude towards the practice. 20 answered that they mainly worked in environments where new functionality was developed. Here,
only one answered that the environment had a negative impact on the attitude towards the practice Planning game. Further, 12 of the respondents stated that their type of environment did not affect the attitude towards the practice, and seven answered that the environment had a positive impact on their attitude towards the practice. Thus, in both types of environments roughly 50% of the respondents answered that their type of environment did not affect their attitude towards the practice Planning game, and 50% stated that the environment had a positive impact on the attitude towards the practice. Only one respondent stated that the environment had a negative impact regarding the attitude towards the practice. Thus, the data does not support the conclusion.

Survey results for questions A1 and C7 were analyzed. A1 was used since it addresses which types of development the teams performed. C7 was used since it addresses whether the teams thought that their type of development affected their attitude towards the practice Planning game. The hypothesis is that a maintenance and support environment reduces meaning with the planning game as it is difficult to know in advance which issues to implement. The Fisher’s Exact test was used. The calculated p value was 0.68. Thus, the null hypothesis was not rejected. This means that the analysis based on these data, does not show that there is a connection between how meaningful the practice Planning game is, and whether the team performed new development of code or maintained and supported code.

7. Incoming support issues during an iteration should be regarded as real issues that belong to the planning game.

Survey results for question C6 were analyzed. C6 was used since it addresses whether the teams thought that incoming support issues during an iteration should be included in the planning game. The hypothesis is that such issues should be managed by the planning game as it provides a better overview and structure for the team how to prioritize their development tasks. The data was analyzed by using descriptive statistics since the purpose was to investigate whether a majority of the survey respondents stated that support issues should be included in the planning game. 16 out of 23 stated that incoming support issues during an iteration should be included in the planning game. Four out of 23 did not have an opinion in the issue, and two out of 23 stated that support issues should not be included in the planning game. Thus, a majority of the answers stated that support issues should be included in the planning game. Further, nine of the respondents stated that they worked in maintenance and support environ-
ments. Out of these, seven stated that incoming support issues should be included in the planning game. 20 of the respondents answered that they worked mainly with development of new functionality, and 15 of these stated that incoming support issues should be included in the planning game. In both types of environments roughly 75% of the respondents answered that incoming support issues during an iteration should be included in the planning game. Thus, the conclusion is supported by the survey participants, regardless of which type of environment XP is used in.

8. It is difficult to introduce XP where all or most of XP’s practices are applied continuously.

The survey results show that the survey participants introduced 136 practices. Out of these, 97 were used continuously. This means that 71% of the practices were introduced and then used continuously. It is interesting to notice that the practice Coding standard was introduced six times and then always used continuously, whereas the practice Metaphor for instance was introduced four times but then never used at a continuous level. A possible explanation for the low number of introductions for these practices may be that the practice Coding standard was commonly used in organizations, even before XP was released. Thus, many organizations did already apply this concept. Regarding the practice Metaphor, the low number may depend on the difficulty of understanding the purpose of the practice and how to apply it. These aspects may also explain why the practice Coding standard was used continuously in all of the cases, whereas the practice Metaphor was never used continuously.

Survey results for questions B3 and D1 were analyzed. B3 was used since it addresses whether any practices were used before the process was introduced. D1 was used since it addresses which practices were introduced and used continuously. The hypothesis is that it is uncommon to introduce a practice and then apply it continuously, since development needs usually change during a software project. The Spearman method was used since the data are measured on an ordinal scale and two variables are compared. The value of Spearman’s rho was 0.53. Thus, as a perfect correlation is either -1 or 1, the value 0.53 suggests a rather weak correlation between the two variables. To test the significance of this relationship, the calculated rho is compared to the critical rho value. The value of 0.53 and 10 degrees of freedom provide a p value that is above the chosen probability error threshold of 0.05. Thus, there was not a statistically significant relationship between the variables. This means that the analysis based on these data,
does not show a strong correlation between introducing a practice and using it continuously.

9. *The introduction of a process based on XP to a team increases communication and knowledge transfer within the team.*

Survey results for question C1 were analyzed. C1 was used since it addresses effects received from introducing the agile method. The hypothesis is that the introduction of processes based on XP increases communication and knowledge transfer within the teams as agile methods are based on communication and openness. The data was analyzed by using descriptive statistics since the purpose was to investigate whether a majority of the survey respondents stated that the introduction of a process based on XP to a team, increased communication and knowledge transfer within the team. 15 out of 23 stated that the introduction of the process had a positive effect on communication, and 18 out of 23 stated that the introduction of the process had a positive effect on knowledge transfer. Thus, a majority of the respondents stated that the introduction of a process based on XP increased communication and knowledge transfer.

This view is also supported when considering that the respondents were of the opinion that the surrounding organization’s attitude towards the team changed to being more positive as a result of introducing the agile method. This change in attitude may depend on the fact that the team had become more competent as a result of increased communication and knowledge transfer. 11 of the respondents stated that the surrounding organization’s attitude towards the team was more positive as a result of the team using the agile method, and only two stated that the agile method had a negative impact on the organization’s attitude.

10. *People within an organization that collaborate with a team using a process based on XP increase their trust to the team.*

Survey results for questions C2, C3 and C4 were analyzed. C2 was used since it addresses to what extent the surrounding organization was aware that the team started using an agile method. C3 was used as it addresses whether the teams changed their way of communicating with the surrounding organization as a result of using an agile method. C4 was used since it addresses whether the teams perceived if and how the surrounding organization’s attitude changed as a result of the teams using agile methods. The hypothesis is that the surrounding organization increases its trust to a team when it starts using an agile method since agile methods help teams improve both their internal
communication, but also externally to surrounding organizations. The data was analyzed by using descriptive statistics since the purpose was to investigate whether a majority of the survey respondents supported the conclusion. 16 out of 23 perceived that the team changed its way to communicate with the surrounding organization. 19 out of 23 were of the opinion that the surrounding organization was aware that the team had started using a process based on XP. Further, 11 out of 23 were of the opinion that the surrounding organization’s attitude toward the team was more positive after the process was introduced than before. Only two out of 23 thought it had a negative effect. Thus, a majority of the respondents stated that the introduction of the agile method had a positive effect on the surrounding organization’s attitude toward the team. Further, 19 of 23 stated that the surrounding organization was aware that an agile method had been introduced. Thus, it is reasonably safe to argue that the surrounding organization increased its trust to the team, as a result of the team starting using the agile method.

Four out of ten conclusions were regarded as reasonably accepted by the organizations when analyzing the results from the survey. However, in these cases simple descriptive statistics were used to determine their validity. In the other cases where the conclusions were not regarded as valid, more advanced approaches using statistical tests were used. Although it is difficult to clearly state that the conclusions are valid by analyzing the results from this survey, it does not necessarily mean that they are not valid generally speaking. In many cases there are probably not enough data to perform this type of analysis. One conclusion concerned how to treat incoming support issues during an iteration. If a team receives many issues during an iteration and does not include them in the planning game, the meaning with having a planning game will be reduced as planned tasks are not conducted and the customer will not know what has been done. This view was supported by the survey participants as they wanted to include support issues in the planning game. Another conclusion concerned the fact that it is difficult to implement the practice *On-site customer* without adaptations, as real customers are seldom available. This view is also supported in several articles. The conclusion is important as XP stresses the need of having a competent customer who drives the development work forward. However, as this result shows real customers are seldom used which surely has a negative impact on XP usage. The two other conclusions that were supported by a majority of the respondents concern positive effects of increased communication which according to these results is a consequence of applying processes based on XP. The rest of the conclusions that were not validated mainly concern in which type of environments processes are introduced and how these environments affect the way processes are introduced. Thus, it is reasonable to
state that processes are introduced in many different environments and it is
difficult to predict what the results of these introductions will be. Further,
it is difficult to state whether the type of development environment has any
effect on the introduction.

12.6 Concluding Remarks

This survey addresses how processes based on XP are introduced, adapted
and used in various software development organizations. There exists little
knowledge how and why these processes are introduced, adapted and used
in different ways. An improved understanding of these activities would help
organizations more efficiently apply XP when developing software. 23 com-
panies participated in the survey and shared their experiences regarding this
issue. The analysis of the collected data generated a number of conclusions.
Below, three conclusions are mentioned which address important aspects
when implementing processes based on XP in organizations.

The first conclusion addresses how processes based on XP are introduced
into organizations. It states that it is difficult to generally classify how
processes are introduced, and what the results of these introductions are in
relation to the types of development environments they are introduced into.
For instance, although many organizations in the survey were alike in the
sense that they developed new code, they differed significantly regarding
which parts of XP that were introduced or applied continuously. A possible
explanation for this is that software development often involves many people
with unique experiences and skills. Further, each organization has its own
rules and culture. All these aspects may affect how and what parts of a
process that are introduced.

The second conclusion addresses how processes based on XP are adapted
in organizations. The results from the survey show that those practices that
the study participants reported most problems with when using them, were
adapted to a larger extent than those where there were few or no problems
with. Thus, when people have problems with using a practice they tend to
adapt it to some extent in order for the practice to be more convenient to
use. Although this approach increases the applicability of the practice, it
may have a negative effect. If a practice is adapted too much it may loose
any benefits it generated when using it as XP advocates.

The third conclusion addresses the usage of processes based on XP. A
majority of the survey participants reported that they already used roughly a
third of XP’s practices in some way before the agile method was introduced.
This fact should be considered when planning SPI initiatives in form of
introducing software development processes into organizations. It may not
be necessary to apply all parts of the new process, since similar activities
may already be used in the organization. Thus, by performing a thorough
review of the current software development process, an organization may realize which parts of a new process that may be useful and which parts that may not.
STUDY 5: A SURVEY ON AGILE DEVELOPMENT
Chapter 13

GUIDELINES FOR INTRODUCING PROCESSES BASED ON XP

13.1 Background

The support for agile methods concerns how to improve the introduction of processes based on XP into organizations. This activity was chosen since there exists a limited amount of support for this activity. As mentioned in Section 3.3, the term support is defined as any kind of approach that either improves product quality and/or reduces costs and development time. Based on the results from Study 4 and Study 5, which contained both qualitative and quantitative data, a number of guidelines were formulated to support the introduction of processes based on XP.

The support for agile methods is not based on data collection and analysis, as is the case for the support for plan driven methods. It is the opinion of the author of this thesis that the developed support should match the characteristics of the actual process type. Thus, since agile development is based on an informal approach to development which should be easy to manage, it was not regarded as appropriate to develop support for agile methods similar to the plan-driven method support.

Study 4 and Study 5 identified potential problems when introducing processes based on XP. These problems are addressed in the guidelines presented below. By considering these guidelines, problems associated with the introduction of processes based on XP may be avoided. There are many aspects to address when introducing a software development process into an organization. A number of them may be regarded as obvious, such as informing and educating the people who will use the new process. Of course, these as-
pects should also be considered and planned for when introducing a process even if they seem obvious. However, education and other issues which may be regarded as obvious are not addressed by these guidelines. The guidelines are based on results and experiences from Study 4 and Study 5. However, they have not been used in a real software development environment. Thus, the guidelines are considered as preliminary and should be incrementally tested and refined under real circumstances.

13.2 Guidelines

The guidelines are presented below in different areas which are considered important to address when introducing a process based on XP. The guidelines are based on either the experience gained when conducting Study 4 or on conclusions from Study 4 and Study 5, or a combination of experience and conclusion. After each guideline follows complementary information that further explains the meaning of the guideline. Thus, by considering these guidelines an organization is better prepared to successfully introduce a process based on XP. The areas are presented in random order. Thus, the guidelines in the first area may not necessarily be more important than the guidelines presented in the last area.

The software development process. The guidelines presented below are applicable when software development organizations assess their processes, before introducing processes based on XP.

- *If an organization has developed a software architecture it is not necessary to introduce the practice Metaphor.* The results show that an already established software architecture makes the use of a metaphor unnecessary. Further, the results show that it is not trivial to define an appropriate metaphor which is understood by everyone and addresses all aspects of a system in an adequately way.

- *Plan well in advance which person or persons in the organization who will act the on-site customer role.* The results show that it is rarely the case that a real external customer will be available to act the on-site customer role. Instead, people working within the organization will have to act the on-site customer role. In these cases, it is important to choose an appropriate person or persons that will have time to participate with the team. Further, the person or persons should have the necessary competence and authority so that the customer role will lead the development, as XP advocates.
Type of development. The guidelines presented below are applicable for software development organizations that work mainly with maintenance and support.

- *Allocate slack time for unplanned incoming software issues in the planning game, if an organization focuses on maintenance and support.* The results from Study 4 showed that the practice *Planning game* reduced its meaning when applied in a maintenance and support context. It became less meaningful to allocate time for planned issues to be implemented during an iteration since new, unplanned issues arrived in frequent random fashion. The team successfully addressed this problem by leaving available time in the planning game, for the issues that would arrive during the iteration. This way the practice became meaningful as there was time available to implement the planned issues. The amount of slack time should be proportional to the amount of support issues that are expected to arrive during an iteration.

- *Introduce the practice Collective code ownership if an organization performs maintenance and support.* The results indicate that the use of the practice *Collective code ownership* increases a team’s level of competence. Further, the results also indicate that software maintenance and support environments facilitate use of the practice *Collective code ownership*, to a higher degree than contexts where focus is on new development. This may depend on the fact that team members are more likely to review more areas of code, for instance when trying to locate a software fault. Thus, the practice should be introduced in maintenance and support environments as it more likely will be applied to a larger extent.

Maintaining several systems. The guideline presented below is applicable for software development organizations that maintain more than one system.

- *Plan for that the amount of time needed to introduce the practice Continuous testing, is proportional to the number of systems a team maintains.* The results show that the practice *Continuous testing* is cumbersome and time-consuming to introduce on one system. The results from Study 4 also show that it is more work associated with introducing the practice on several systems. This depends on the fact that an unit test framework should be installed on each system. Further, the systems may be implemented in different languages and on different platforms which reduces the possibility of reusing code from other unit test frameworks.
Economical issues. The guidelines presented below are applicable when software development organizations consider economical aspects when introducing processes based on XP.

- **Try to avoid the situation where members of the team that will use a process based on XP are financially allocated to different projects.** There may be people such as project managers who "own" people in form of so called resources in different projects. This means that a resource (i.e. person) is devoting his/her total effort to services associated to a specific project. This concept of allocating resources may be problematic if applied in an XP project, since XP assumes a close collaboration between team members. Thus, the collaboration within an XP team may be reduced if team members financially belong to different projects.

- **Make sure that management is committed to the introduction of XP.** It is important not to underestimate the effort and time needed to introduce processes based on XP properly. Although XP belongs to the agile methods, which by many people are considered to be easy to introduce, it takes effort and time to introduce the process properly. Thus, it is important for a team that will start using XP to have commitment from management so that the team can spend time on introducing and adapting the process, instead of working with the daily tasks.

These guidelines provide support to organizations in their efforts of introducing processes based on XP. One aspect that is not included in the guidelines is the physical location of the team members, since it is regarded as an obvious aspect to consider when introducing processes based on XP. However, it is our opinion that it has such a profound effect on the results of applying XP that it is worth mentioning. The results from Study 4 show the importance of having team members physically close to each other in order to facilitate communication within the team. Thus, it is important to assess how a team is physically located when a process based on XP is introduced into an organization. XP advocates that a team is placed in the same room. In Study 4, the team members were placed in adjacent rooms which were regarded as sufficiently close. Further, in most cases there were at least two team members in a room. However, it is our opinion that the team members should not be placed further away from each other than adjacent rooms, since it is a risk otherwise that the informal and ad hoc communication which characterizes agile development may otherwise be lost. Thus, it is important to clarify how a team will be physically located when a process based on XP is introduced and make sure that the team members are placed sufficiently close to each other.
13.2. GUIDELINES

When a team has assessed its development environment it is in a better position to understand what parts of XP that should be introduced and how these parts may be adapted. For instance, one important decision is whether to introduce parts of XP at a time or all of XP at once. The results from Study 4 and Study 5 place no valuation that one approach is better than another. However, it is important to introduce XP in a pace that is manageable by both the team and its surrounding organization. Further, both organizational and team aspects should be considered in order to provide optimum support when introducing a process based on XP into an organization.

The team that used the agile method, described in Study 4.1 to Study 4.3, was interviewed after the case study had ended and asked for its opinion how the introduction activity could improve. This is important as a better understanding of how to introduce processes based on XP more likely leads to successful implementation of XP in a long term view. The team mentioned three aspects, that it had recognized afterwards would likely improve the introduction of the agile method.

- **Support from management** to introduce XP, that will make the introduction of the agile method a high priority task.

- **No reorganizations** during the introduction of the agile method. Reorganizations may have a negative impact on the team. If the team decreases in number or if people are exchanged between teams can the team, at least temporarily, lose competence.

- **New development.** XP should be used when a team’s main activity is new development instead of focus on maintenance and support. For example, new development of functionality makes the planning game activity easier which was considered by the team in Study 4 to be an important tool for planning and controlling software development.

Software development organizations can use these preliminary guidelines as examples of important aspects to consider when introducing processes based on XP. The results from Study 4 and Study 5 show that it is a complex task to introduce processes based on XP and an increased understanding of this issue may help organizations more often successfully introduce this kind of process.
Part IV

Epilogue
Chapter 14

SUMMARY OF CONTRIBUTIONS

14.1 Overview

This chapter presents the contributions of the thesis. First, an overview is provided regarding the thesis’ types of contributions. This is followed by a summary of the conclusions from the studies. Further, an analysis regarding the validity of the results is provided. The chapter ends with a discussion of future research opportunities.

14.2 Types of Contributions

The thesis’ contributions can be divided into different types where each type helps address the overall research problem of developing support for agile and plan-driven methods. Below, the contribution types are presented followed by motivations why these contributions help address the research problem.

Empirical data. The support for agile and plan-driven methods are developed based on a thorough empirical basis. Both qualitative and quantitative data were collected and analyzed. The empirical data provide examples of problems and solutions when developing support for agile and plan-driven methods.

Support for plan-driven methods. The developed support for plan-driven methods is based on collection and analysis of process data. It provides means for individuals to improve their work in software development contexts. The contribution is two-fold. First, support is developed for the Business Process Analyst role to manage its work in a software development context. Further, the PSP is allocated to a number of roles in order to increase its applicability.
Support for agile methods. The developed support for agile methods consists of a set of guidelines which can provide support when introducing processes based on XP into organizations. Guidelines is considered an appropriate form to provide support for this kind of activity since they identify important issues to address. Further, they also provide individual freedom when applying the guidelines which is important since each organization has its unique culture and needs.

A survey on introducing processes based on XP. Study 4.1 to Study 5 provide descriptions how and in what types of environments processes based on XP are introduced into. Further the results address how these processes are adapted and used, and why. This type of contribution provides referenced knowledge regarding the introduction of processes based on XP. Thus, these studies provide examples how processes based on XP may be introduced and adapted, which are useful for organizations to consider which are interested in XP.

The developed support is based on an extensive amount of empirical data which increases the validity of the support. The studies address the overall research problem in different ways. Study 1 provides experience how to collect and analyze process data when using a plan-driven method (i.e. PSP) which was useful when conducting Study 2 and Study 3. Study 2 transfers the type of support that PSP provides into a new context. Study 3 presents a solution how to increase the applicability of PSP by allocating its contents to several roles instead of one role. Study 4 investigates the results from introducing a process based on XP into a complex software development context. Further, the development environment is described in detail and the results are based both on qualitative and quantitative data. In Study 5, a survey is conducted where data is collected regarding organizations’ use of processes based on XP. Further, the survey addresses how these processes were introduced and adapted. The guidelines presented in Section 13.2 are based on the results from Study 4 and Study 5.

14.3 Conclusions from the Studies

This section provides a summary of the studies described in Chapters 5 to 7 and in Chapters 9 to 12, their main conclusions and how the overall research problem of developing support for agile and plan-driven methods is addressed in each study.

14.3.1 Study 1: Code Review Efficiency

This study addresses how code reviews at an individual level can be improved. The study investigates to what extent the three criteria mentioned
in Chapter 5: 1) the level of detail of a code review checklist, 2) the update of the checklist, and 3) the number of LOC reviewed per hour affect the efficiency of code reviews. These criteria have been identified by Humphrey [41] to affect code review efficiency, but it has not been investigated what their order of precedence is regarding positive impact on code reviews. The research question is formulated as:

*How does the application of the criteria: 1) level of detail, 2) update and 3) LOC/hour affect code review efficiency?*

The results indicate that the criterion Update provided most positive impact on code review efficiency, followed by the criterion Level of detail and the criterion with the least impact was LOC/hour. Further, the results show that conducting design and code reviews save about 22% in defect removal time, compared to if defect removal activities only consist of compile and test activities. Further, the results show that it is not necessary with major SPI initiatives to improve software development. On the contrary, significant improvements in software development performance can be achieved with minor efforts. In this case through the use of code reviews. This is an important insight since it is our experience that many major SPI initiatives do not prevail due to various reasons, such as an increased demand of communication in an organization where people already have busy schedules. The conclusion from the study is:

**Conclusion from Study 1:** The criterion *The update of the checklist with a new defect profile* has more positive impact on code review efficiency than the criterion *The level of detail of the checklist*, which in turn has more positive impact than the criterion *The number of reviewed LOC/Hour*.

The purpose of the study with regard to the overall research problem was to gain experience of collecting and analyzing data when using a plan-driven method (i.e. PSP). This experience was useful as it helped understand the importance of developing support that is both meaningful and applicable. Although this viewpoint may be regarded as trivial, it is not a trivial task to develop meaningful and applicable support. To do this, it is important to develop support based on empirical results in order for the support to be valid in its context.

### 14.3.2 Study 2: SPI Support Based on Non-Technical Criteria

This study addresses to which extent the application of non-technical criteria can provide SPI support in a software development context. Non-technical criteria are usually difficult to quantify and thus control. There exist some
support to manage non-technical criteria in other environments, for instance provided by Lientz and Rea [60] and Johansson [48]. However, the underlying reason for conducting this study is that it is assumed to be more difficult to provide SPI support when applying non-technical criteria in a software development context. The research question is formulated as:

*To which extent can the application of non-technical criteria provide SPI support in a software development context?*

The support was developed in form of a framework containing a number of templates, metrics and guidelines to apply when managing various non-technical criteria in a software development context. The framework was evaluated by people from the software development community that worked with non-technical criteria, and the average result from the evaluation was 3.3 out of 5 where 5 indicates total support for managing and improving non-technical criteria. Thus, the conclusion from the study is:

**Conclusion from Study 2:** The application of non-technical criteria can provide adequate SPI support in a software development context.

The underlying assumption for this study was that it is difficult to provide SPI support based on the application of non-technical criteria in software development environments. This assumption is based on the fact that the characteristics of these environments involve tight time schedules and complex activities where people participate in different projects. These aspects can make it difficult to manage non-technical criteria since the collection of such kind of data is not automatic and takes time. However, as the results from the study show it is possible to provide SPI support in a software development context through the application of non-technical criteria.

Thus, an important conclusion from Study 2 that concerns support for plan-driven methods is that it is possible to provide support based on PSP in different contexts than the context which PSP supports, even if these areas may involve non-technical criteria that can be difficult to measure and control. This is an important conclusion since many roles involved in software development deal with non-technical criteria and if support can be provided for managing these roles, the overall software development performance of an organization may increase. Thus, SPI support may not only originate from activities which tend to be measurable such as programming and testing (which PSP supports) but also from non-technical criteria in a software development context.

### 14.3.3 Study 3: Individual SPI Support in RUP

This study addresses how a process at an organizational level (i.e. RUP) may receive SPI support at an individual level. RUP consists of a number of
roles. However, these roles do not have support how to improve their work routines. The roles who receive SPI support at an individual level are likely to improve their work performance, since they have the means to collect and analyze process data regarding their work. This may in turn improve the overall performance in the actual software development project they are involved in. The research question is formulated as:

*How can SPI support at an individual level be introduced into RUP?*

The contents of PSP (i.e. phases, logs, templates and metrics) were allocated to a number of roles in RUP. Further, guidelines how to introduce, adapt and maintain the contents of PSP in a RUP context were provided. These guidelines are important since every project and organization is unique, and the incorporation of PSP into RUP will have to be adapted to the specific needs of the current project and organization. The conclusion from the study is:

**Conclusion from Study 3:** The structure and content of RUP are suited for SPI support at an individual level in form of PSP.

The study’s problem regarding developing support for plan-driven methods concerns the low applicability of PSP. It is the opinion of the author of this thesis that PSP is not used in the industry, since it is too cumbersome to apply due to its many process elements (e.g. metrics and templates). This study presents a solution how the applicability of PSP may be increased. This is done by allocating PSP to several roles instead of one, thus reducing the number of metrics and templates a role has to manage.

**14.3.4 Study 4.1: Quantitative Measures on Agile Development**

This study reports quantitative results when introducing an agile process based on XP to an evolutionary and maintenance software development environment. There is a lack of empirical studies regarding what results to expect when using agile methods, which this study addresses. The research question is formulated as:

*Does the introduction of an agile process to an evolutionary and maintenance software development environment improve software development performance?*

The results from this study are based on a quantitative analysis regarding a team’s performance while using a process based on XP. The results do not support the notion that agile development increases a team’s performance.
For example, the analysis of the data did not show a significant improvement of the team’s productivity after compared to before the agile method was introduced to the team. Thus, the conclusion is:

**Conclusion from Study 4.1:** The introduction of a process based on XP to a software development team does not necessarily imply a significant improvement in the team’s performance.

### 14.3.5 Study 4.2: Qualitative Measures on Agile Development

This study addresses a team’s experiences regarding how a process based on XP was introduced and adapted. Further, it investigates how an evolutionary and maintenance software development environment affects the introduction and adaptation of XP’s practices. The results from this study are based on a qualitative analysis regarding how the process was adapted and how the team’s development environment affected the introduction of the process. The research question is formulated as:

What are the experiences of introducing an agile process based on XP to an evolutionary and maintenance software development environment?

The results show the importance of performing an evaluation of a team’s development environment before an agile process based on XP is introduced, in order to identify what parts of the process that are needed by the team and which parts that are not. Although the conclusion is nontrivial in the sense that it originated from formal analysis of the results from the case study, it appeals to common sense. An introduction of a process into an organization without any thought on how to adapt it seems unlikely. Thus, the conclusion is:

**Conclusion from Study 4.2:** An organization’s software development environment should be assessed before introducing a process based on XP, in order to minimize the efforts of introducing the process and optimizing the results of using the process.

### 14.3.6 Study 4.3: Agile Collaboration Between Team and Organization

This study addresses how collaboration between a software development team and its surrounding organization is affected during introduction of a process based on XP. The results are based on the opinion of the surrounding organization, as it is important to include the experiences of the organization when introducing a software development process in order for it to be adapted optimally. The research question is formulated as:
How does the use of an agile process affect the collaboration in a software development organization?

The results of this study are based on both a qualitative and quantitative analysis of collected data. The results show that the surrounding organization perceived that its collaboration with the team that used the agile method improved as a result of introducing the process. This result supports the notion that agile development facilitates communication within an organization. Thus, the conclusion is:

**Conclusion from Study 4.3:** The introduction of a process based on XP to a software development team is likely to have a positive effect on the collaboration between the team and its surrounding organization.

The developed support for agile methods was chosen to address the introduction of processes based on XP. The case study generated a number of results, seen from two perspectives (i.e. the team and its surrounding organization). Further, the results originated both from qualitative and quantitative methods. Thus, the case study provides a balanced picture of the introduction of a process based on XP into an organization which helps develop both meaningful and applicable support. Further, the case study provides a substantial amount of empirical data on which the support (i.e. the guidelines) was developed.

### 14.3.7 Study 5: A Survey on Agile Development

This study investigates Scandinavian organizations’ experiences of introducing agile methods based on XP. The case study presented in Study 4 gathered information locally regarding the results of introducing an agile method based on XP. The purpose of this study is to gain information gathered from many organizations, thus increasing the validity of the results. Thus, to complement the result from Study 4 a survey was conducted in this study in which 23 software development organizations participated. The research question is formulated as:

What are the results of introducing agile methods based on XP into software development organizations?

The organizations where asked a number of different questions which were believed to affect the introduction and use of processes based on XP such as the motivation for introducing the method, how it was introduced (gradually or all at once) or how the surrounding organization affected a team’s use of a process based on XP. In Study 4, the organization that was introduced to the agile method was large and complex where several systems were maintained and supported. A motivated question to ask is whether
there exists some connection between how a process based on XP may be
introduced efficiently into an organization, with regard to the characteristics
of that organization. The survey covered a broad spectrum of organizations
that differed in many ways, such as type of development and complexity.
However, after analyzing the collected data it was not possible to detect any
connections between how to introduce processes based on XP versus the
characteristics of the organizations. Thus, the conclusion from the study is:

**Conclusion from Study 5:** There is no connection between how processes
based on XP are introduced into organizations and the characteristics
of these organizations.

Study 5 generated more knowledge regarding the introduction of pro-
cesses based on XP to complement the results from Study 4. Further, some
of the results from Study 4 were supported by the results from Study 5.
Thus, the validity of these results is increased which in turn increases the
validity of the developed support. The support for agile methods (i.e. the
guidelines) was based on the results from Study 4 and Study 5.

### 14.4 Analysis of Validity

Robson [84] classifies threats to validity in two types. The first type, internal
validity, concerns whether the treatment actually caused the outcome of the
study. Internal validity may in turn be divided into a number of threats.
One of these is time, that is if something occurred during the study that
affected the outcome of the study. The time aspect was not a threat when
developing support for plan-driven methods. However, in Study 4 there were
reorganizations during the case study which meant that one person in the
team was removed from the team. Further, the unstable economic climate
may have had a negative effect on the team that applied the agile method
since the team members were concerned about loosing their jobs. Another
aspect that applied to the research in the thesis is so called diffusion of
treatments, which means that several groups are chosen where each group
receives a special treatment in order to investigate what the effects are from
each treatment. In Study 2, one role was chosen to receive support based
on PSP. Preferably, two or more roles should have received support in order
to establish which kind of context that was most suitable to receive such
support. However, due to time limitations this option was not chosen. The
same line of reasoning holds for Study 4. Preferably two teams would have
been measured in parallel where one team used a process based on XP and
the other team developed software as it use to. However, this option was
not chosen since there did not exist another team that worked solely with
a system. This is important since it is vital that results originate from that
team only in order to investigate the outcome of the treatment.
The second type, external validity, concerns if the results are generalizable to other settings. The result from Study 2 (i.e. the developed framework) was assessed by BPAs from the industry. Thus, although the framework has not been used under real circumstances it has been evaluated by experienced people from the industry. The allocation of PSP into RUP, described in Study 3, is a theoretical approach how the applicability of PSP may be increased. The ambition was not to evaluate it in the industry. Thus, little can be said about how the presented solution applies in different contexts. Obviously, the results from the case study in Study 4 may only be valid to the unique environment where the study took place. However, by conducting the survey in Study 5 the generalizability of the results was investigated.

Another aspect that affects how generalizable research results are is how the sample data is defined and collected since the sample data should adequately represent the target population. One important aspect of this issue is the number of study participants. In Study 2, seven people participated in the study. More people may have increased the generalizability of the results, but we did not manage to find any more people that worked as BPAs. However, the study participants had various backgrounds and experiences thus increasing the likelihood that the sample represented a typical BPA. In Study 5, 23 organizations participated in the survey. It is difficult to state whether this may be regarded as an adequately large number, since we do not know how many organizations that have introduced processes based on XP. However, similar to Study 2 the sample data differed as the organizations that participated in the survey were of different types and complexity. Thus, although the number of study participants may be low, the diversity of the sample data increases the likelihood that it adequately represents the target population.

14.5 Future Work

This section discusses future research opportunities for the studies included in the thesis. Below, suggestions regarding future work for each study is provided.

**Study 1: Code Review Efficiency.** The purpose of Study 1 was to gain experience of collecting and analyzing data when using PSP. Future work may include the collection and analysis of process data when using other plan-driven methods than PSP, since this experience is important when developing support for plan-driven methods.

**Study 2: SPI Support Based on Non-Technical Criteria.** Although the results indicate that the application of non-technical criteria can
provide adequate SPI support in a software development context, further evidence is needed as the framework has not yet been evaluated in practical use. Thus, future research includes introducing the framework into industrial organizations where it can be evaluated in real environments.

**Study 3: Individual SPI Support in RUP.** The value of this work is a theoretical approach to provide workers in RUP with a process for improving their individual software engineering skills. However, this needs to be investigated in practice. A study should be conducted monitoring problems and results of applying PSP improvement concepts in real RUP projects. PSP research (e.g. [93], [41] and [71]) show that software engineers applying PSP improve their software engineering processes, but the question is if these positive benefits still exist after adapting and disseminating PSP’s elements to a number of roles.

**Study 4: A Case Study on Agile Development.** The results of Study 4 identified a number of problems when introducing a process based on XP into a complex software development environment. It would be interesting to perform a new case study where XP is introduced into a similar development environment where these problems are addressed with appropriate actions. The results from that case study could be compared to the results from Study 4, in order to evaluate how much more efficient the introduction was in the latter case study.

**Study 5: A Survey on Agile Development.** A major conclusion from Study 5 is that there is no connection between how processes based on XP are introduced into organizations, and the characteristics of these organizations. A future research opportunity would be to investigate the validity of the conclusion, since if the conclusion is not valid it is important to identify what aspects in an organization that affect how processes based on XP should be introduced.

In the thesis we have made a number of claims regarding support for agile and plan-driven methods. These claims address different contexts when managing agile and plan-driven methods, such as managing non-technical criteria in a software development context. However, the results (i.e. SPI framework, theoretical analysis and guidelines) have not been used under real conditions. Thus, it is important to evaluate their usefulness and meaningfulness in the industry and to test, update and refine the results as more response regarding their use is received.
Appendix A

TERMS AND CONCEPTS USED IN SPI

The terms presented below have the following semantics in the thesis:

*Framework*
A framework can be seen as an environment defined for a purpose, which supports activities performed in that environment.

*Model*
A model is a simplified representation of a system or phenomenon by extracting, from the modeler’s perspective, important characteristics.

*Process*
A process is a set of interrelated activities, which transform inputs into outputs.

*Software Process Improvement*
The activity of understanding existing software development processes and changing these processes to improve product quality and/or reduce costs and development time.

*Standard*
A standard encapsulates best practice to perform an activity. It provides rules, guidelines or characteristics for activities and their results, aimed at optimum achievement in a given context.

*Technique*
A practical skill or art applied to a particular task.

*Software Process Maturity*
The extend to which a specific process is explicitly defined, managed,
controlled and effective. Maturity implies a potential growth in capability and indicates both the richness of an organization’s software process and the consistency with which it is applied in projects throughout the organization.

**Software Process Maturity Level**

Defines the route to attaining full maturity in software development. The purpose of applying software maturity levels is to complete the software development process.

**Process Data**

Representation of information of a process in a formalized manner suitable for communication, interpretation or processing by automatic means.
Bibliography


[80] Reed, J. F. Solutions to the Behrens–Fisher Problem. *Computer Methods and Programs in Biomedicine* 70, 3 (March 2003), 259–263.


