ESPOO 2006 VTT PUBLICATIONS 618

Outi Salo

Enabling Software Process Improvement in Agile Software Development Teams and Organisations

VTT PUBLICATIONS 618

Enabling Software Process Improvement in Agile Software Development Teams and Organisations

Outi Salo

Academic Dissertation to be presented with the assent of the Faculty of Science, University of Oulu, for public discussion in the Auditorium L10, Linnanmaa, on January 12th, 2007, at noon.

ISBN 951–38–6869–9 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6870–2 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455–0849 (URL: http://www.vtt.fi/publications/index.jsp)

Copyright © VTT Technical Research Centre of Finland 2006

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
Puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
Tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 3, P.O.Box 1000, FI-02044 VTT, Finland, phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT, Kaitoväylä 1, PL 1100, 90571 OULU
Puh. vaihde 020 722 111, faksi 020 722 2320

VTT, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
Tel. växel 020 722 111, fax 020 722 2320

VTT Technical Research Centre of Finland, Kaitoväylä 1, P.O. Box 1100, FI-90571 OULU, Finland, phone internat. +358 20 722 111, fax +358 20 722 2320

Edita Prima Oy, Helsinki 2006

Salo, Outi. Enabling Software Process Improvement in Agile Software Development Teams and Organisations. Espoo 2006. VTT Publications 618. 149 p. + app. 96 p.

Keywords software process improvement, SPI, agile software development, iterative improvement process

Abstract

Agile software development has challenged the traditional ways of delivering software as it provides a very different approach to software development. In recent decades, software process improvement (SPI) has been widely studied in the context of traditional software development, and its strengths and weaknesses have been recognised. As organisations increasingly adopt agile software development methodologies to be used alongside traditional methodologies, new challenges and opportunities for SPI are also emerging. One challenge is that traditional SPI methods often emphasise the continuous improvement of organisational software development processes, whereas the principles of agile software development focus on iterative adaptation and improvement of the activities of individual software development teams to increase effectiveness.

The focus of this thesis is twofold. The first goal is to study how agile software development teams can conduct SPI, according to the values, principles and practices of agile software development, in tandem with the success factors of traditional SPI. The second goal is to study how the team-centred SPI of agile software development and the traditional view of organisational improvement can be integrated to co-exist in a mutually-beneficial manner in software development organisations. The main research methodology in this thesis is action research (AR). The empirical data is taken from six agile software development case projects. The results of this research have been published in a total of seven conference, and journal, papers.

The principal result of the study of project level SPI is an Iterative Improvement Process which provides systematic, yet agile, SPI mechanisms for agile software development teams. This process iteratively evolved during the series of case

projects. The empirical evidence of the project level research demonstrates the ability and willingness of agile software development teams to iteratively improve their daily working practices by making minor and simple, yet effective and visible, improvements during their projects. The research data further indicates the positive effect of iterative team reflection on the satisfaction of project teams, and confirms the need for systematic mechanisms to carry out SPI activities in agile project teams. Furthermore, the data shows that external support for the improvement activities proved to be highly significant for the success of SPI within agile project teams.

The study of organizational SPI initially focused on integrating agile software development and continuous improvement of existing organisational practices. Arising from this stage, several changes to traditional SPI activities were suggested in order to establish a mutually-beneficial co-existence between organisational SPI stakeholders and agile software development teams. During the research a framework for deploying agile practices in organisations was developed. In this novel framework, the Iterative Improvement Process provides a mechanism for feedback between the agile software development teams and continuous organisational improvement activities. The research data further indicates that documented and validated knowledge arising from the Iterative Improvement Processes of agile software development teams can be beneficial in other contexts, such as in analysing and establishing future SPI initiatives in software development organisations.

Preface

My interest in software process improvement (SPI) developed in the late 1990s when I was doing my master's thesis while working as a System Designer in Elbit Oy. Thus, firstly, I would like to thank my former employer for providing me with such an interesting topic of research that still interests me. In both my doctoral and my master's theses, I have been privileged to have Professor Samuli Saukkonen from the University of Oulu as my supervisor. In addition to his supervision, I am also grateful for his encouragement to pursue a career in research. I would like to express my sincere gratitude to Professor Pekka Abrahamsson – my supervisor at VTT (VTT Technical Research Centre of Finland) – for his contribution in supporting and setting the scene for my research.

At VTT, I have been provided with an ideal environment for conducting my research. In 2002, VTT launched the UUTE project (New Software Technologies) to examine agile software development methodologies, which was where I started the research for this thesis. TEKES (National Technology Agency of Finland) has been an important funder of the ICAROS (Integrating End-User Needs and Business Competence to Mobile Software Development Concepts) and Agile ITEA I and II projects (Agile Software Development of Embedded Systems) where this research has been carried out. I would like to sincerely thank all the support personnel of VTT for making our research work so much easier. On the language front, I wish to thank language consultants Seppo Keränen and Hilary Keller for their work on my thesis.

Furthermore, I am truly grateful for the very fruitful discussions, valuable feedback, and support from my fellow-researchers at VTT, especially the members of the agile research projects, former OTE research group, as well as the Process Innovations and Mobile Services and Applications research teams with whom I have had the great privilege to work with during the course of this study. My special thanks to Annukka Mäntyniemi and Minna Pikkarainen for their friendship and encouragement throughout. I would also like to mention Matias Vierimaa, Dr. Seija Komi-Sirviö and Dr. Tua Huomo who have all encouraged me to finish this work and allocated me time in which to do it. This study would not have been possible without the enthusiasm and teamwork of the

members of the support and quality teams of ENERGI (Industry-Driven Experimental Software Engineering Initiative) in all the SPI activities undertaken, many thanks to Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Mikko Korkala, Juha Koskela, and Pekka Kyllönen. The management, customers and especially the software developers in all the case projects have been the core of this research and I am very grateful for your collaboration.

The manuscript of this thesis was reviewed by Professor Brian Fitzgerald, of the University of Limerick in Ireland, and Professor Giancarlo Succi, of the Free University of Bolzano-Bozen, Italy. Their extremely constructive comments have greatly improved the final outcome.

Lastly, but most dearly, I would like to express my gratitude to my parents Kerttu and Lauri Hakala for all their support and encouragement down through the years, to my husband Petri for his love and patience, to my beloved children, Joonatan and Samuel, for being an endless source of joy and inspiration, to my dear sister, Kaisu Hakala, for her true friendship, and my mother-in-law Valma Salo. I have also been very lucky to have been blessed with great and loyal friends in my life. You have all put up with me through the times of stress, shared the joys and sorrows of life with me, encouraged me onwards, helped me to survive the hectic everyday life, and have had faith in me to get this work completed. Without your existence and support, this just would not have been possible. Thank you from the bottom of my heart.

Mäntylä, Oulu, Finland, November 2006.

Outi Salo

"Why worry, there should be laughter after pain
There should be sunshine after rain
These things have always been the same
So why worry now?"

- Mark Knopfler -

List of Original Publications

The following publications have been included in this thesis. They have all been published either in the proceedings of international conferences with appropriate review processes, or in scientific journals.

I Salo, O. & Abrahamsson, P. Empirical Evaluation of Agile Software Development: the Controlled Case Study Approach. PROFES 2004, 5th International Conference on Product Focused Software Process Improvement, Keihanna-Plaza, Kansai Science City, Kyoto-Nara, Japan, April, 2004.

II Salo, O., Kolehmainen, K., Kyllönen, P., Löthman, J., Salmijärvi, S. & Abrahamsson, P. Self-Adaptability of Agile Software Process: A Case Study on Post-Iteration Workshops. XP 2004, 5th International Conference on eXtreme Programming and Agile Process in Software Engineering, Garmisch-Partenkirchen, Germany, June, 2004.

III Salo, O. Improving Software Process in Agile Software Development Projects: Results from two XP Case Studies. 30th EUROMICRO Conference, Rennes, France, August, 2004.

IV Salo, O. Systematical Validation of Learning in Agile Software Development Environment. LSO 2005, 7th International Workshop on Learning Software Organisations, Kaiserslautern, Germany, April, 2005.

V Salo, O. & Abrahamsson, P. An Iterative Improvement Process for Agile Software Development. *Software Process Improvement and Practice*. In press to be published in Vol. 12, Issue 1, 2007. Published online 6th October, 2006 (http://www3.interscience.wiley.com).

VI Salo, O. & Abrahamsson, P. Integrating Agile Software Development and Software Process Improvement: a Longitudinal Case Study. ISESE 2005, 4th International Symposium on Empirical Software Engineering, Noosa Heads, Australia, November, 2005.

VII Pikkarainen, M., Salo, O. & Still, J. Deploying Agile Practices in Organisations: A Case Study. EuroSPI 2005, European Software Process Improvement and Innovation Conference, Budapest, Hungary, November, 2005.

Hereafter, the papers will be referred to by their Roman numerals.

List of Names and Acronyms

AL	Action Learning
APM	Agile Project Management
AR	Action Research
AS	Action Science
ASD	Adaptive Software Development
CMM®	Capability Maturity Model®
CMMI®	Capability Maturity Model Integration®
CSF	Critical Success Factor
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
GQM	Goal-Question-Metric method
EF	Experience Factory
ENERGI	Industry-Driven Experimental Software Engineering Initiative
IID	Iterative and Incremental Development
IS	Information Systems
KM	Knowledge Management
OSSP	Organisation's Standard Software Process
PAR	Participatory Action Research
PIW	Post-Iteration Workshop method
pm	Person Month
PMA	Post Mortem Analysis

QIP	Quality Improvement Paradigm
ROI	Return on Investment
RUP	Rational Unified Process
SEI	Software Engineering Institute
SEPG	Software Engineering Process Group
SPA	Software Process Assessment
SPC	Statistical Process Control
SDP	Software Development Plan
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability Determination (ISO 15504)
TDD	Test-Driven Development
TSP	Team Software Process℠
TQC	Total Quality Control
VTT	Technical Research Centre of Finland (i.e., Valtion Teknillinen Tutkimuskeskus)
XP	Extreme Programming

Contents

Abstract 3

Preface 5

List of Original Publications 7

List of Names and Acronyms 8

1. Introduction 13
   1.1 Background 13
   1.2 Focus of Research 14
   1.3 Research Problem 15
   1.4 Outline of the Thesis 15

2. Related Work 17
   2.1 Process Models of Software Development 17
      2.1.1 Plan-Driven Models for Software Development 18
      2.1.2 Iterative Change-Driven Models for Software Development 21
   2.2 Agile Software Development 24
      2.2.1 History and Fundamentals of Agile Software Development 24
      2.2.2 Current Status of Agile Software Development 27
   2.3 Software Process Improvement 28
      2.3.1 Traditional Elements of SPI 29
         2.3.1.1 Organisational SPI Models 30
         2.3.1.2 Standard Processes and Assessments 32
         2.3.1.3 Process Tailoring 33
         2.3.1.4 Process Deployment 36
         2.3.1.5 Measurement 37
         2.3.1.6 Experience, Knowledge and Learning 38

3. SPI in Agile Software Development 41
   3.1 The Elements of SPI in Agile Software Development 41
         3.1.1.1 Organisational Models 41
         3.1.1.2 Standard Processes and Assessments 42
         3.1.1.3 Process Tailoring 44
         3.1.1.4 Process Deployment 53

3.1.1.5 Measurement 55
3.1.1.6 Experience, Knowledge and Learning 56
3.2 Comparison of SPI Elements in Plan-Driven and Agile Software Development 57

4. Research Design 66
4.1 Research Methods and Evolution 66
4.1.1 Literature Review 70
4.1.2 Case Study 71
4.1.3 Action Research 71
4.1.3.1 Conceptual Foundation of AR 74
4.1.3.2 Study Design of AR 76
4.1.3.3 Research process of AR 78
4.1.3.4 Role Expectations of AR 79
4.1.4 Five Cycles of Action Research 81
4.1.4.1 Diagnosing 83
4.1.4.2 Action Planning 85
4.1.4.3 Action Taking 86
4.1.4.4 Evaluating 88
4.1.4.5 Specifying Learning 88
4.2 Research Setting 90
4.2.1 Context of Research 90
4.2.1.1 Laboratory Research Setting 90
4.2.1.2 Semi-Industrial Research Setting 92
4.2.1.3 Industrial Context 93
4.2.2 Case Projects and Organisation 94
4.2.2.1 Case Project I: eXpert 98
4.2.2.2 Case Project II: zOmbie 99
4.2.2.3 Case Project III: bAmbie 100
4.2.2.4 Case Project IV: uniCorn 101
4.2.2.5 Case Project V: Bubble 101
4.2.2.6 Case Project VI: Phantom 102
4.3 Collection of the Empirical Evidence 103
4.4 Storing and Analysing the Empirical Evidence 105
4.5 Reporting the Results of Research 108
4.5.1 Paper I 109
4.5.2 Paper II 109

4.5.3 Paper III 110
4.5.4 Paper IV 111
4.5.5 Paper V 112
4.5.6 Paper VI 112
4.5.7 Paper VII 113
4.6 Summary 113

5. Evaluation of the Research 115
5.1 Validity of Research 115
5.2 Evaluation of the Results 118
5.2.1 Implications for the Theory 119
5.2.2 Implications for the Practice 125

6. Summary and Conclusions 129
6.1 Summary of the Results 129
6.2 Limitations of the Study 130
6.3 Future Research 132

References 134

Appendices

Papers I–VII

1. Introduction

In the introduction, the background of the research, its focus, research questions and the structure of the thesis are presented.

1.1 Background

Software process improvement (SPI) has been extensively studied in the past few decades. In traditional SPI methods and approaches (e.g., Basili 1994, Basili & Caldiera 1994, SEI 2001), the aspect of organisational improvement has usually been placed in a central role, due to the fact that the planning and control of the SPI initiatives are managed by the organisational stakeholders. The reported positive effects of SPI methods and approaches include reducing time-to-market, risks and costs, and increasing the productivity and quality in software development organisations (Krasner 1999, van Solingen & Berghout 1999). However, various negative effects have also been encountered, e.g. regarding the cost-effectiveness of SPI initiatives, their actual effectiveness in improving the software development practices of organisations, the volume of the effort needed to implement SPI initiatives and the low speed at which visible and concrete results are achieved (Dybå 2000, Goldensen & Herbsleb 1995, Krasner 1999). In fact, it has been reported that around two-thirds of SPI traditional initiatives fail to achieve their intended goals (Debou 1999).

From the mid-1990s onwards, agile software development principles and methodologies have been increasingly challenging the traditional view of software development. A Forrester Study (Schwaber & Fichera 2005) indicates that 14% of North American and European companies are already using agile software development processes, and 19% are interested in adopting them or planning to do so. In addition, the study reports that while the interest in, and awareness of, agile software development is increasing, so is the confusion about what it really means to “go agile” (Schwaber & Fichera 2005, p. 1).

The values and principles of the agile manifesto (Agile Alliance 2001) identify the central elements of agility that should be embedded in any method claiming to be agile. Although the research and methodologies of SPI are still limited in agile software development, SPI has been given a central role in the agile

manifesto principles (Agile Alliance 2001). One of the twelve principles states that agile teams should regularly reflect on their work in order to become more effective, while another principle addresses the self-organisation of teams. Thus, the fundamentals of agile software development address the improvement and management of software development practices within individual teams. Agile software development provides a highly untraditional approach to SPI, in which the process improvement knowledge of software developers and software development teams is acknowledged and valued. Currently, however, while numerous agile software development methodologies encourage project teams to carry out regular reflection, the means for doing so seems to be undefined.

Agile software development provides new possibilities for conducting SPI, which may well provide grounds for meeting some of the central challenges of traditional SPI.

1.2 Focus of Research

Two specific perspectives can be identified in this research at project level and at organisational level. Firstly, the project level SPI of this research examines the regular process adaptation activities in individual agile software development teams, emphasised in the principles of agile software development. In this respect, this thesis aims to identify how process adaptation can be conducted in a manner that complies with the fundamentals of agile software development (Agile Alliance 2001) while also fulfilling the criteria of successful SPI methods (Komi-Sirviö 2004).

Secondly, from the organisational viewpoint, this research focuses on how the regular SPI activities of individual agile project teams can be integrated into traditional organisational SPI mechanisms. In this respect, this thesis examines how agile software development teams, focusing on the immediate improvement of their daily working practices, can co-operate in a mutually-beneficial manner with the organisational SPI stakeholders who, in turn, are concerned with the continuous improvement of organisational software processes. Another organisational perspective in this study is to examine how the process adaptation of agile software development teams can benefit the organisational deployment of agile software development methodologies.

1.3 Research Problem

Due to the explanatory nature of the case studies of the research, the form of the research questions addresses the aspect of "how" (Yin 2003). The literature review, conducted at the beginning of the research, revealed the importance of SPI, visible also in the agile principles (Agile Alliance 2001) in their encouragement for regular team reflection of agile software development teams in order for them to become more effective. Although such SPI activities had been addressed especially on the project level of agile software development, a very limited amount of empirical evidence or methods had been put forward. Therefore, the first research problem is:

Q.1. How to conduct SPI in individual agile software development teams?

In the SPI literature, the central and underlying tenet has traditionally been the continuous improvement of organisational processes in software development organisations (Zahran 1998). The literature review, however, revealed that this aspect of SPI principle has not been considered in agile software development. Thus, this study had to examine how traditional SPI, grounded on organisational improvement, and the iterative and ongoing improvement of individual agile software development teams could be adjusted and integrated to co-exist in a mutually-beneficial manner in software development organisations. Therefore, the second research problem is:

Q.2. How to integrate the agile SPI activities of individual project teams into traditional continuous organisational SPI activities?

In this thesis, this question includes the related aspects of continuous improvement of existing organisational practices, and the deployment of agile methodologies in organisations.

1.4 Outline of the Thesis

The focus of this research is defined in section 1 of the thesis, in section 2 the related work is discussed. The related work consists of two main areas: 1) the models of software development (section 2.1) including the so-called plan-

driven and iterative approaches, and 2) traditional SPI elements (section 2.3). In this thesis, there are six main elements identified in the core of SPI: organisational models of SPI, standard processes and assessments, process tailoring, process deployment, measurement, and also experience, knowledge and learning. Agile software development is defined as a part of iterative software development and is defined in more detail in section 2.2. The traditional mechanisms of SPI are discussed mainly to provide a background and reference to discuss the beliefs and the differences between SPI in agile and traditional software development. However, as the traditional SPI mechanisms are not the direct focus of this study, nor have they been been applied in the research as such, a thorough analysis of traditional SPI mechanisms and their practical implementation has been left out of the discussion.

In section 3, there is a discussion of how the traditional elements of SPI have currently been addressed in the context of agile software development. In addition, the differences between SPI in the contexts of traditional and agile software development are summarised and discussed.

In section 4, the research design is described and it presents the different phases of the case studies as identified by Yin (2003). In section 4.1, the different research approaches and methods of the research are defined, i.e., literature review (i.e., preliminary study), case study (i.e., case project I), and AR (Action Research) (i.e., case projects II–VI). As a central part of this research, the application of AR characteristics – as identified by Lau (1999) – is described in section 4.1.3, and the application of the five high-level cycles of AR – as identified by Susman and Evered (1978) – is presented in section 4.1.4. The research setting is defined in section 4.2 and sets out the context and organisation of the research. In addition, the collection, storage and analysis of the empirical evidence, as well as the reporting of the results of the research (i.e., publication process of Papers I–VII) are addressed in the research design section of this thesis.

In section 5, the validity of the research and the evaluation of the research results are discussed. Section 6. presents a summary of the research results, the limitations of the research and outlines future research avenues. The seven original papers of this research are included as the final part of the thesis.

2. Related Work

In this section, the literature related to the topic of this thesis is reviewed. Firstly, in order to situate the position of agile software development among the other software development approaches, sub-section 2.1 includes a discussion of the evolution of the central process models used in software development. In this thesis, the software development models are divided into traditional, plan-driven models and iterative, change-driven models. In addition, the background and fundamentals of agile software development are defined in section 2.2. In section 2.3.1, the different elements of SPI are discussed, as defined in the context of traditional software development.

2.1 Process Models of Software Development

The primary function of software development process models is to "determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to the next" (Boehm 1988, p. 61). During the history of software development, different models and approaches have been suggested for tackling the complexity and uncertainty of software development. Figure 1 illustrates the evolution of process models in the past decades. As can be seen in the y-axis of Figure 1, it has also been suggested that the evolution of software development models originates from the problems of ad hoc programming that, at first, led towards traditional plan-driven models and towards iterative change-driven models of software development. The original meaning of the Latin term 'ad hoc' refers to a methodology that has been designed for a special purpose (ad hoc = 'for the purpose of'). However, in this context – as often in software engineering literature (e.g., Basili & Reiter 1981) – the term 'ad hoc' is used to refer to the low degree of methodological discipline. It should also be noted that the positioning of the different software development models on the y-axis in Figure 1 is illustrative rather than scientific.

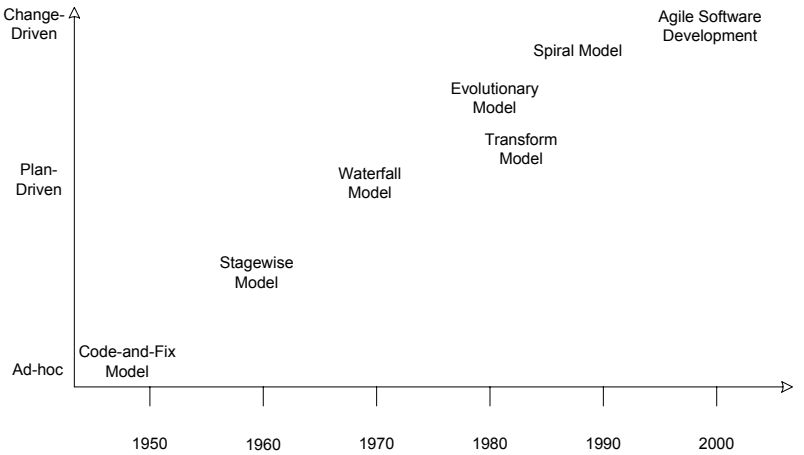

Figure 1. The Evolution of Software Process Models.

In the following sub-sections, the evolution of software development process models is discussed in more detail.

2.1.1 Plan-Driven Models for Software Development

The plan-driven approaches of software development have been defined as document-driven, code-driven, and traditional process models (Boehm 1988). As the names suggest, a common feature for the plan-driven process models is their emphasis on defining the scope, schedule, and costs of the project upfront including, for example, an early fixing stage and extensive documentation of the end product requirements. One common characteristic could also be the recurrence of the software development phases only once during the development process, i.e., with only hints of iterativity (Larman & Basili 2003). In the following sections of this thesis, the process models of this category will be referred to as traditional software development.

The two-step process model of code-and-fix, used in the early days of software development, resulted in difficulties that necessitated explicit sequencing of the phases of software development (Boehm 1988). In particular, the need to design prior to coding, to define requirements prior to design, and the need for early preparation for testing and modification were identified (Boehm 1988). One of the first models to rise to that challenge was the stagewise model as early as in

the middle of the 1950s (Benington 1983). This model evolved from the problems caused by the increasing size of software programs, which could not be handled by a single programmer (Benington 1983).

In 1968, the NATO Science Committee held a software engineering conference in Garmisch, Germany, where the "software crisis", or "software gap", was discussed (NATO Science Committee 1969). A standardisation of the software development process with an emphasis on quality, costs, and development practices was the key recommendation of the conference (Lycett et al. 2003). Soon after this, as a refinement of the stepwise model, the waterfall model was introduced. The early version of the waterfall model was introduced in 1970 by Royce (1970) and it has since evolved into a concept consisting of the sequential phases of requirements analysis, design, and development (Larman & Basili 2003). According to Boehm (1988), the waterfall model provided two main advances over the stepwise model: it introduced prototyping to parallel the stages of requirements analysis and design, and provided feedback loops between the sequential stages. It should also be noted that, already in the early waterfall model of Royce (1970), it had been realised that it might be necessary to first build a pilot model of the system, i.e., to conduct two cycles of development and to obtain feedback to adjust the model. Thus, hints of iterativity in the model can be seen yet "this iterative feedback-based step has been lost in most descriptions of this model, although it is clearly not classic IID" (Larman & Basili 2003, p. 48). Today, the waterfall model has been adopted for most software acquisition standards in government and industry (Boehm 1988). While the waterfall model has solved various core problems in software development, it also includes features not appropriate for every software development context (Boehm 1988). One central problem of the waterfall model has been identified as its "emphasis on fully elaborated documents as completion criteria for early requirements and design phases" (Boehm 1988, p. 63).

The V-Model can be considered a variation of the waterfall model. The original V-Model includes similar phases to the waterfall model but its phases are not defined as a linear activity but form a V-shape. The V-Model first became a standard for German civil and military federal agencies in 1997, as a result of the Development Standards for IT Systems of the Federal Republic of Germany. In this model, the Coding- phase is situated in the intersection of the V, while the

software design – software verification, system design – system verification, and requirements engineering – system validation form the crescent counterparts each side of the V-shape. The model emphasises traceability between the requirements, design and implementation. The latter version of the V-Model, i.e. V-Modell® XT (KBSt – Federal Government Co-Ordination and Advisory Agency 2004), however, has been extended to cover the entire system life-cycle and aims to be compatible with standards such as CMMI and to increase the scalability and adaptability of the model. In addition, the later version of the V-Model also perceives the possibility of conducting a series of subsequent V-cycles which increases the possibility of applying the model in a more iterative manner.

More commonly, it has been argued, that "no life-cycle scheme, even with variations, can be applied to all system development" (McCracken & Jackson 1982, p. 30). On the other hand, according to the survey study of Fitzgerald (1998), despite numerous existing software development methodologies, as much as 60% of software development organizations do not apply any development methodologies. An additional problem has been identified in using a disciplined approach to software development which is that, "rather than focusing on the *end* (the development of software), developers become pre-occupied with the *means* (the software development method)" (Fitzgerald et al. 2004, p. 65). In practice, the result may be the disparity between the organisational software development process and its actual implementation in the software development teams (Fitzgerald 1997).

Another dilemma identified among plan-driven approaches to software development, is the pursuit of certainty. The up-front requirements definition, and locking of the project scope, leads to contracts and decisions based on estimations of costs, time and resources. However, such estimates have been found to be highly prone to uncertainty (Morien 2005). Nonetheless, the success of software projects is often measured against these estimates as it may be appealing, from the viewpoint of both an acquirer and supplier, to agree fixed costs, scope and schedule for the project up-front. However, it has been stated that "certainty is a myth and is the most uncertain part of any project" (Morien 2005, p. 519). In fact, it could be argued that the quest for certainty, in both time and money, may not only fail to pay off in these respects but may also seriously affect the quality of the end product.

It can be argued that the plan-driven models of software development can and should be applied in a dynamic way by repeating the phases or even the entire process, if necessary. However, the original purpose of these process models was not to welcome changes during the development, but rather to try to fix factors, such as scope, time and money, up-front in order to eliminate change which was considered a risk factor.

2.1.2 Iterative Change-Driven Models for Software Development

The central software development models, developed after the waterfall model, seem to have the common aim of enabling, at least to some degree, the evolution of product requirements during the process of software development. This contributed one main modification to the earlier software development models: the adoption of the iterative and incremental approach. Iterative development refers to the overall lifecycle model in which the software is built in several iterations in sequence (Larman 2004). According to Larman, each iteration can be considered as a mini-project in which the activities of requirements analysis, design, implementation and testing are conducted in order to produce a subset of the final system, often resulting in internal iteration release. An iteration release has been defined as "a stable, integrated and tested partially complete system" (Larman 2004, p. 10). Incremental development involves adding functionality to a system over several releases, i.e., a repeated delivery of a system into the market or production. Thus, one incremental delivery may be composed of several iterations. A development approach where the system is developed in several iterations is called iterative and incremental development (IID), yet it is often referred to as iterative development. (Larman 2004)

Even though agile software development has recently brought the IID approach of developing software into the spotlight, the history of these approaches is, in fact, considerably longer (Larman & Basili 2003). Many of the earlier change-driven approaches have adopted the ideologies of prototyping, for example, where the first early prototype gradually evolves into the final software product with no formal specifications or co-operation with the customer (McCracken & Jackson 1982). Among the first models that focused on increasing the possibility of determining product improvements throughout the development process, was the evolutionary development (Evo) model. This concept was first introduced in

1981 (Gilb 1981) and has been expanded by Gilb (1988, 2005). This method suggested an iterative development approach in which the product increment was understood as a delivery to the real customer rather than a prototype (Gilb 1981). While evolutionary delivery also lacks plans for future deliveries, it does attempt to capture feedback to guide future deliveries. This is in contrast to "pure" incremental delivery where the plan is drafted for several future deliveries and feedback is not the sole driving force (Larman 2004).

The evolutionary model was followed by the transform model (Balzer et al. 1983), which is also based on the iterative development model and on adjusting the product during the development. The transform model, however, had a strong emphasis on product specifications due to its ideology of focusing on automatic transformation of specifications into code (Boehm 1988). This approach had its origin in the problems of the earlier software development models producing "spaghetti code", which was difficult to modify and maintain (Boehm 1988).

The spiral model of the late 1980s (Boehm 1988) typically consists of four iteratively repeatable steps: 1) determining the objectives, alternatives, and constraints, 2) evaluating alternatives, and identifying and resolving risks, 3) development and verification, and 4) planning the next phase. Boehm (1988) defined the spiral model as a risk-driven approach for software development. In the spiral model, the iteratively evaluated strategy for resolving the risks of the next spiral has an effect on the choice of the software development approaches to be adopted. Depending on the risks, the spiral model then allows the adoption of any mixture of development approached, such as prototyping or elements from the specification-oriented waterfall approach modified to incremental development. According to Boehm, the risk-driven approach also means that the results of each risk analysis activity has an effect on the amount of time and effort allocated to the different development activities in the following spiral, while also influencing the required level of completeness, formality, or granularity of product specifications. (Boehm 1988)

Agile software development, which emerged in the mid-1990s, can also be classified as an iterative and change-driven software development approach. It could be argued that at present there is no common agile process model with specified phases, but there is rather a set of fundamentals (Agile Alliance 2001)

common to the methods claiming to be agile. However, Extreme Programming (XP) (Beck 2000), which is probably the best-known among the first agile methodologies, contains an underlying process model for agile software development that has been adopted and adapted by its successors. Figure 2 illustrates how Beck (1999) has compared the agile development model of XP with the waterfall model and with the iterative processes.

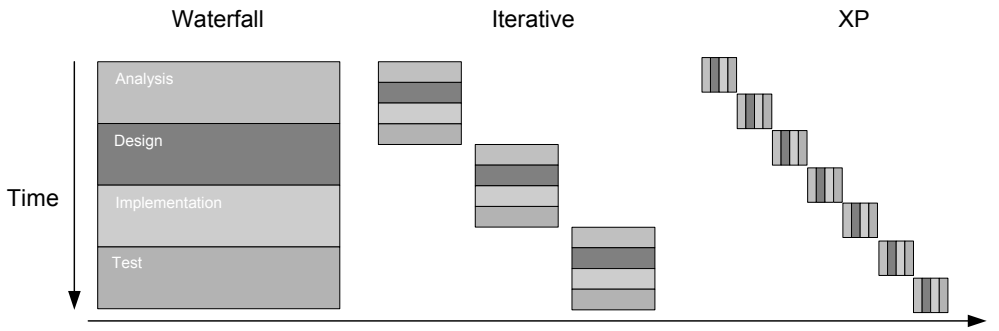

Figure 2. Process Models in Comparison (Beck 1999).

The simplified illustration of the different software development models (Figure 2) provides an overview of the suggested differences between the models. According to Beck (1999), XP aims at blending the activities of analysis, design, implementation and testing, a little at a time, throughout the entire software development process. The common feature of agile methods is the recognition that software development cannot be considered to be a defined process, but rather an empirical (or nonlinear) one due to the constant changes that are welcomed during the development of the software product (Williams & Cockburn 2003).

According to Larman, "in modern iterative methods, the recommended length of one iteration is between one and six weeks", (Larman 2004, p. 11) whereas the "incremental deliveries are often between three and twelve months" (Larman 2004, p. 20). The principles of agile development suggest a short (i.e., from two weeks to two months) duration of the development iterations. Evo also promotes relatively short delivery cycles of few weeks (Larman 2004). Similarly as in the evolutionary model, agile methods also consider the term "iterative" as referring to evolutionary advancement of the product rather than just rework (Larman & Basili 2003).

Agile software development is discussed in more detail in section 2.2.

2.2 Agile Software Development

In this section, the background, fundamentals and current status of agile software development are discussed.

2.2.1 History and Fundamentals of Agile Software Development

The emergence of agile methodologies can be said to have begun in the mid-1990s, when software methodologies and techniques such as Extreme Programming (XP) (Beck 1999), Scrum (Schwaber 1995), eXtreme testing (Jeffries 1999), Crystal Family of Methodologies (Cockburn 1998), Dynamic Systems Development Method (DSDM) (Stapleton 2003), Adaptive Software Development (ASD) (Highsmith 2000), and Feature-Driven Development (FDD) (Coad et al. 1999) began to emerge. The emergence of agile methodologies is defined in more detail in, for example, (Abrahamsson et al. 2002, Abrahamsson et al. 2003).

The ideologies of agile software development can be traced back to lean manufacturing in the 1940s as well as agile manufacturing in the early 1990s. Lean manufacturing is based on the fundamentals of short-cycle time, reduced setup, multi-skilling and flow being in place while driving out waste in time, activity, inventory and space (Ross & Francis 2003). The essence of the agile approach in manufacturing has been summarised as "the ability of an enterprise to thrive in an environment of rapid and unpredictable change" (Gould 1997, p. 28). While the debate between the actual differences of lean and agile is still going on in the manufacturing sector (e.g., James 2005), the central ideologies of both can be found in the fundamentals and methodologies of agile software development. For example, in Lean Software Development (Poppendieck & Poppendieck 2003) the lean principles are integrated with agile practices.

In software development, the agile "movement" was launched in 2001 when the various originators and practitioners of these methodologies met to identify the common aspects of these methods that both combined old and new ideas, and

clearly shared some particular ideologies in common. As a result, the Manifesto for Agile Software Development was drafted and the term "agile" was chosen to combine the methods and techniques that would share the values and principles of agile software development. The values and principles of the Agile Manifesto (Agile Alliance 2001) set out the central elements of agility that should be embedded in any method claiming to be agile. The agile manifesto emphasises the agile values listed below on the left, while the items listed below on the right are still considered valuable too:

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

The twelve principles of agile software development (Agile Alliance 2001) are: 1) the highest priority is to satisfy the customer through early and continuous delivery of valuable software, 2) the welcoming of changing requirements, even late in development, for the benefit of the customer's competitive advantage, 3) frequent delivery of working software, the release cycle ranging from a couple of weeks to a couple of months, with a preference for a shorter timescale, 4) daily collaboration of business people and developers throughout the project, 5) building of projects around motivated individuals by offering them an appropriate environment and the support they need, and trusting them to get the job done, 6) emphasis on face-to-face conversation for conveying information and within a development team, 7) working software is the primary measure of progress, 8) agile processes promote a sustainable development pace for the sponsors, developers, and users, 9) continuous attention to technical excellence and good design enhances agility, 10) simplicity is essential for maximising the amount of work not having to be done, 11) self-organising teams give best results in terms of architectures, requirements, and designs, 12) regular reflection of teams on how to become more effective, and tuning and adjusting its behaviour accordingly. The principles of agile software development can be considered as fundamental ideologies that should be embedded in the practices of any software development method claiming to be agile.

The core features of agility that should be embedded in any true agile method have been further specified as follows: iterative development of several cycles, incremental development, ability and permittance of teams to self-organise and determine the management of work, and emergence of processes, principles, and work structures during the project (Boehm & Turner 2003a). In addition, the active involvement of users in requirements and planning, and the importance of tacit knowledge are identified as further important elements of agile software development (Boehm & Turner 2003a).

Essentially, many of the ideologies behind the agile software development methods are not – nor have they been claimed to be – new. Many of these ideologies and related agile software development methodologies have roots in, for example, the preceeding iterative methodologies (Abrahamsson et al. 2003) and agile and lean industrial product development (Poppendieck & Poppendieck 2003). In addition, it has been widely acknowledged prior to the agile movement that the different methods of software development are far from being neutral and universally applicable (Malouin & Landry 1983). Benington, among many others, has earlier considered top-down programming and specification as highly misleading and dangerous, as it assumes that enough detailed knowledge is available up-front to precisely know the objectives before producing a single line of code, and because it erroneously parallels the software development to the manufacturing industry (Benington 1983). Furthermore, the positive effect of regular employee involvement in operating decisions and a high degree of responsibility for overall performance in high team spirit, loyalty, and motivation have also already been recognised among production workers (Deming 1990). Neither has the iterative or incremental mode of software development been invented only by agile proponents, but it has a long history in software development (see Larman & Basili 2003). However, the agile software development approach has accomplished a novel mixture of old and new software development principles that have been gaining increasing interest among practitioners and researchers alike. Williams and Cockburn suggest that the novelty of agile software development is, "if anything, the bundling of the techniques into a theoretical and practical framework" (Williams & Cockburn 2003, p. 40).

In conclusion, the fundamentals of agile software development propose a very different view to the certainty aspect in the software development process,

compared to the plan-driven approaches (see 2.1.1.). In agile software development, the uncertainty of schedule, scope and budget of any software development project can be considered as an baseline assumption. Thus, agile software development methodologies can be regarded as a means of responding to the uncertainty of software development, rather than as a means of achieving certainty.

2.2.2 Current Status of Agile Software Development

Currently, there is considerable discussion in scientific forums both in favour and against agile methodologies. The early agile methodologies, especially, received criticism for the lack of scientific evidence (Abrahamsson et al. 2002, Lindvall et al. 2002), and their suitability only for software development contexts where small and co-located teams were producing non-safety-critical products with volatile requirements (Williams & Cockburn 2003).

Since the early days of agile software development, an increasing amount of interest has been paid to agile methods, by both practitioners and researchers, thus creating a growing body of empirical data on the different aspects of agile software development. Apart from the individual methods and practices of agile software development, problematic issues have arisen, such as the scalability of agile software development for large and multisite projects (e.g., Eckstein 2004, Lindvall et al. 2004) and the compatibility of agile methods with existing standards (Lycett et al. 2003, Paulk 2001, Reifer 2003). Recently, the organisational and business aspects of agility have been receiving more attention (e.g., Baskerville et al. 2005, Coplien & Harrison 2005, Oleson 1998). Accordingly, the early agile methods and techniques have been evolving and are being updated – e.g., XP (Beck & Andres 2004), Scrum (Schwaber 2004, Schwaber & Beedle 2002), Crystal (Cockburn 2005), Test-Driven Development (TDD) (Beck 2003), and DSDM (DSDMConsortium 2003).

Currently, as more empirical evidence on the agile methodologies is available, it seems that the main arguments for and against their use is nowadays not so much about their benefits, but rather about the need to extend their scope and adapt them to organisations with established and mature plan-driven processes (e.g., Boehm & Turner 2005). For instance, it has been suggested that one major

problem in adopting agile methodologies can be found in balancing the currently dominating engineering ideologies and methodologies of manageable, predictable and repeatable processes with agile software development methods, which again embrace self-organisation, process adaptation and constant changes (Table 5) (Lycett et al. 2003). Balancing the two approaches has been suggested in order to benefit from their strengths, and to compensate for their weaknesses (Boehm & Turner 2003b).

In addition, there has been some confusion regarding the relationship between ad hoc coding and agile software development. It has been proposed that one reason for this confusion is the piecemeal approach of agile software development (Highsmith & Cockburn 2001). For instance, quality in design in agile software development is prioritised in ongoing design done in smaller chunks instead of massive up-front design of the system (Highsmith & Cockburn 2001). In fact, the existing agile methodologies, such as Scrum for agile project management and XP for implementation of software, all seem to propose a rather disciplined approach to conducting the tasks of software development. In addition, studies (e.g., Kähkönen & Abrahamsson 2004, Nawrocki et al. 2001, Paulk 2001) indicate that by adopting different agile methods and practices, individual agile software development teams can accomplish a methodology that meets with the goals of CMMI level 2. However, there still seems to be a need to extend agile methodologies in order to meet, for example, CMMI requirements related to more organisational level practices.

2.3 Software Process Improvement

A software process can be defined as "the sequence of steps required to develop or maintain software" (Humphrey 1995, p. 4), aiming at providing the "technical and management framework for applying methods, tools, and people to the software task" (Humphrey 1995, p. 5). However, even the most exquisitely defined and managed process may still not meet the context specific needs and objectives of software development organisations and customers regarding, for example, performance, stability, compliance and capability (Florac et al. 2000). Thus, Software Process Improvement (SPI) aims at providing software development organisations with mechanisms for evaluating their existing

processes, identifying possibilities for improving as well as implementing and evaluating the impact of improvements (Florac et al. 2000).

In this section, firstly, the different SPI mechanisms of traditional software development (2.3.1) are discussed. Thereafter, there is a discussion on how SPI is currently addressed in the context of agile software development (3). Representing the central focus of this research, the emphasis of section 3 is on the current process tailoring mechanisms of agile software development (3.1.1.3) and how they are in line with the critical success factors (Komi-Sirviö 2004) of traditional SPI. In the conclusions section, there is a discussion on how SPI in traditional and agile software engineering relate to each other.

2.3.1 Traditional Elements of SPI

Traditionally, the ultimate goal of SPI in organisations is to provide a Return on Investment (ROI) for the organisation through the improvement activities yielding more money than is spent on them (Rico 2004). ROI has been reported for various SPI achievements, such as improved efficiency of the development process and reduction of total software costs, increased quality of the end product, higher predictability of cost and schedule, and increased level of reuse (Krasner 1999). The focus on quality in SPI is based on the fundamental ideology that quality-driven development is likely to yield "not only better quality but also lower cost and improvement of competitive position" (Deming 1990, p. 181).

One of the characteristics of SPI, as traditionally defined, is its emphasis on the continuous improvement of organisational software development processes in terms of performance, stability, compliance, and capability, for instance. Often the existing SPI methods and approaches seem to enhance the underlying business goals and needs in the improvement of organisational software development processes. Florac, for instance, identifies one of the SPI objectives as planning, justifying, and implementing SPI actions that modify the processes to better meet the business needs (Florac et al. 2000). Traditionally, SPI initiatives are also strictly controlled and managed by the organisational stakeholders (Boehm & Turner 2003b).

In this thesis, six different elements are identified in the context of traditional SPI: the organisational models of continuous improvement, standard processes and assessments, tailoring, deployment, measurement, and the utilization of knowledge and learning in SPI. In the following sub-sections, these aspects of SPI are discussed in more detail.

2.3.1.1 Organisational SPI Models

A number of different improvement frameworks are used to support continuous, top-down SPI in organisations, such as Quality Improvement Paradigm (QIP) (Basili 1989), IDEAL^{SM} (McFeeley 1996), and ISO/IEC 15504 Part 7 (ISO/IEC 1998). These frameworks provide organisational procedures for conducting SPI initiatives in a cyclical and ongoing fashion.

QIP was introduced to provide software development organisations with process-focused mechanisms to improve the quality and productivity of software (Basili 1989). The QIP framework originates from the quality improvement paradigms of the manufacturing industry (Basili & Caldiera 1995), such as the Shewart-Deming cycle (Deming 1990) and its derivative Total Quality Control (TQC) (Feigenbaum 1991, Ishikawa 1985). These methods aim at providing "an effective system for integrating the quality-development, quality-maintenance, and quality-improvement efforts of the various groups in an organisation, so as to enable marketing, engineering, production, and service at the most economical levels which allow for full customer satisfaction" (Feigenbaum 1991, p. 6). In the QIP model, the cycles of corporate and project learning are identified. The corporate level activities of improvement include the definition of current status, setting of goals and scheme of improvement, and analysis and storing of experiences and feedback resulting from the project learning cycle, in which the SPI activities of implementing and piloting the improvements in practice and metrics data collection have taken place. In QIP, the reuse of experiences, utilisation of measurements and learning are in a central role (Basili & Caldiera 1995).

IDEAL^{SM} is an SPI model that defines the steps for planning, conducting, and managing SPI in organisations (McFeeley 1996). The model consists of five phases, all of which include different activities for guiding the systematic improvement approach. Originally, the IDEAL model was a life-cycle model for

SPI based upon the standard process of the Capability Maturity Model® (CMM®) (Paulk et al. 1993), which has since been revised for broader application (Gremba & Myers 1997) as in QIP.

In Table 1, the phases of the different organisational SPI models are aligned with the original Shewart-Deming cycle of improvement originating from a manufacturing industry (Deming 1990).

Table 1. Organisational Models of SPI and Comparison of Phases.

<table>
<tr><th>Shewart-Deming Cycle</th><th>QIP</th><th>IDEAL^{SM}</th><th>ISO/IEC 15504</th></tr>
<tr><td rowspan="3">1. Plan</td><td>1. Characterise and understand</td><td>1. Initiating</td><td>1. Examine organisation's needs</td></tr>
<tr><td>2. Set goals</td><td>2. Diagnosing</td><td rowspan="2">2. Initiate process improvement</td></tr>
<tr><td>3. Choose processes methods, techniques, and tools</td><td>3. Establishing</td></tr>
<tr><td rowspan="3">2. Do</td><td rowspan="3">4. Execute</td><td rowspan="3">4. Acting</td><td>3. Prepare and conduct process assessment</td></tr>
<tr><td>4. Analyse results and derive action plan</td></tr>
<tr><td>5. Implement improvements</td></tr>
<tr><td>3. Check</td><td>5. Execute</td><td>5. Acting</td><td>6. Confirm improvements</td></tr>
<tr><td rowspan="2">4. Act</td><td rowspan="2">6. Analyse
7. Package and store experience</td><td rowspan="2">6. Leveraging</td><td>7. Sustain improvement gains</td></tr>
<tr><td>8. Monitor performance</td></tr>
</table>

As Table 1 illustrates, the organisational improvement models as such do not demand or provide any specific SPI methods for conducting different SPI activities but rather suggest the steps to be taken to achieve continuous improvement, thus serving as a roadmap for improvement in organisations. All the models seem to cover the three key SPI objectives defined by Florac et al. (2000): 1) understand the characteristics of existing processes and the factors that affect process capability, 2) plan, justify, and implement actions that will modify the processes so as to better meet business needs, and 3) assess the impacts and benefits gained, and compare these to the costs of changes made to the process. The different approaches, however, address the different overall SPI procedures used within an improvement cycle. For instance, the ISO/IEC 15504

Part 7 bases (phase 3.) the improvement cycle on conducting process assessments (ISO/IEC 1998), and QIP on the reuse and storing of experience and measurements. However, the common feature for all the SPI management models is the setting of improvement goals and needs at the organisational level. The necessity of statistical methodologies has also been commonly addressed in the paradigms of process improvement (Basili & Weiss 1984, Deming 1990, Kuntzmann-Combelles et al. 1992).

2.3.1.2 Standard Processes and Assessments

There are various standard process models, such as CMM® (Paulk et al. 1993), CMMI® (SEI 2001), ISO 15504, i.e., SPICE (Software Process Improvement and Capability Determination), Trillium (Bell Canada 1994), and Bootstrap (Kuvaja & Bicego 1993, Kuvaja et al. 1994) that provide a reference process model against which organisational processes can be assessed and improved. Standard software development process models provide a top-down approach for SPI which offer a framework against which the organisation can evaluate and improve its own processes and identify practices that would increase the maturity of the current processes (SEI 2001).

CMM® was originally developed in 1987 by the Software Engineering Institute (SEI) and it has been the de Facto standard especially in the United States (Krasner 1999). CMM® has since 2000 evolved into the CMMI® model that, in addition to its original five stage model designed for evaluatíng the maturity of an organisation as a whole, includes a continuous model for evaluating the capability of individual process areas of an organisation. Thus, one goal of transferring CMM® into CMMI® was to make it compatible with the ISO 15504 reference model. The CMM® model was originally developed to enable a capability evaluation of suppliers. In other words, a subcontractor would be requested to submit a software process assessment (SPA) in order to guarantee an adequate maturity in its software development processes and to find out how these needed to be improved. However, SPA may also serve company internal SPI purposes by providing general guidelines about where to start improvements, and in which order (Briand et al. 1999). Often the standard process models also provide specific methods, such as Scampi in CMMI® (Ahern et al. 2005), for conducting an appraisal (Iversen et al. 2002). The process assessments may be carried out among the first activities of an

improvement cycle (Table 1) and they often serve as the kick-off for an SPI program, which then focuses on the weak processes identified in the appraisal (Iversen et al. 2002).

It has been claimed that no single assessment model or method can provide the best alternative for all situations, but each of them serve different purposes and have their own strengths and weaknesses (Nielsen & Pries-Heje 2002). For example, the standard assessments are claimed to require a substantial amount of cost and effort from the participants of the software development organisation involved in the different stages of assessment, and also from the external assessors who usually conduct the assessment (Andersen et al. 2002). Consequently, lighter assessment methods have been proposed for identifying the weaknesses of the current software processes. For example, the so-called problem diagnosis method (Mathiassen et al. 2002) "aims to assist SPI groups in identifying software problems and solutions based on the opinions and insights of key organisational actors" (Iversen et al. 2002, p. 164). Other existing lightweight SPA methods are, for example, RAPID (Rapid Assessment for Process Improvement for Software Development) (Rout et al. 2000), EPA (Express Process Appraisal) (Wilkie et al. 2005), and MARES (Anacleto et al. 2004). The selection of an appropriate assessment method can be supported by existing frameworks for assessment strategy selection as suggested in (Nielsen & Pries-Heje 2002).

2.3.1.3 Process Tailoring

The traditional software development approaches have often been criticised for lacking criteria for their applicability and, thus, for being proposed as universally applicable (Malouin & Landry 1983). It has, however, been realised that no such universally applicable methods exist and that it would be erroneous to presume that some method could be transferred with equal success to another context (Malouin & Landry 1983). The same problem applies not only between organisations but also within the different projects of one organisation. While it has become "widely accepted that the methods should be tailored to the actual needs of the development context" (Fitzgerald et al. 2004, p. 66), it has also been realized that the methods need a "significant amount of modification to suit individual development projects" (Kiely & Fizgerald 2005, p. 1).

Some process tailoring approaches can be identified within the traditional SPI context. Firstly, there are the organisational SPI models, such as QIP and IDEAL, providing procedures for continuously tailoring the standard software process (OSSP) of the organisation (Ginsberg & Quinn 1995) in a top-down manner. In this approach, the focus is on tailoring the standard software development processes for an organisation on the basis of organisational business goals and needs. Such OSSP has been stated to "express requirements that all projects' software development processes must meet" (Ginsberg & Quinn 1995, p. 29) within an organisation. Traditional SPI methods for metrics collection and process assessment can, then, be considered as mechanisms used to support the process-tailoring activities. The reference process for tailoring often originates from the existing standard process models. However, the standard process model of CMM, for one, identifies the need for tailoring its key practices before they can be applied in a specific context. For example, the CMM tailoring framework aims at supporting organisations in defining a CMM-based OSSP to fit the organisational needs. It bases its continuous cycle of tailoring on the IDEAL SPI model. (Ginsberg & Quinn 1995).

Another traditional approach for process tailoring is the definition of the project specific software process. It has been suggested that it is done either up-front, while included as a part of a project plan, or conducted as dynamic tailoring when needed during the software development project (Fitzgerald et al. 2004). However, the practical guidance to developers is still very limited on how to approach tailoring (Fitzgerald et al. 2004). In the context of the CMM process standard, it is suggested that project level tailoring is based on the existing OSSP's and organisational tailoring guidelines (Ginsberg & Quinn 1995). The tailoring guidelines have been defined to include "the means by which the organization recognizes the project's responsibility to address the impact of project-specific needs in the project's defined software process" (Ginsberg & Quinn 1995, p. 29). In the CMM tailoring framework (Ginsberg & Quinn 1995), the project-specific tailoring is said to be controlled by the OSSP tailoring guidelines, in which "the process elements, the tailorable attributes for each element, the range for each attribute, and the considerations for selecting a particular range" (Ginsberg & Quinn 1995, p. 33) are defined. It is suggested that such project-specific OSSP tailoring is done consistently by applying the same tailoring guidelines across all projects and basing the tailoring on project-specific needs. Such tailoring guidelines have been identified as "essential for

risk management and overall project success" (Fitzgerald et al. 2004, p. 70). It has also been argued that "once an organization reaches the point where it can identify the various characteristics or contingencies that occur in its development projects, then it is possible to build flexibility into the method, along with the rules to allow developers to identify appropriate choices" (Fitzgerald et al. 2004, p. 70). The sources for defining tailoring guidelines have been proposed, for example, in the previous project records, the results of project post-mortems and brainstorming sessions, surveys and feedback from customers, and participant observations (Fitzgerald et al. 2004). The compilation of this information in database format or process maps is regarded as valuable to enable constant review and augmentation as the organisational practices of software development evolve (Fitzgerald et al. 2004).

Basili and Rombach have further suggested "a methodology for improving the software process by tailoring it to the specific project goals and environment" (Basili & Rombach 1987). The methodology considers sound tailoring to require a characterization of project goals and the environment where the project is to take place, along with characterising the effect that the candidate methods and tools should have on achieving the set goals in that specific environment. Basili and Rombach have suggested the characterisation takes place in a quantitative manner and by analysing large amounts of data to choose the methods and tools to fit the project specific goals and the specific environment. An automated tool called TAME (Tailoring A Measurement Environment) has been proposed to support such quantitative selection of appropriate methods and tools and to tailor them to the needs of the project and organisation (Basili & Rombach 1987, Oivo & Basili 1992). The advantage of the suggested approach is the ability to utilise data from previous (similar) projects in tailoring the practices of future projects. The disadvantage, on the other hand, is that the approach measures the impact of different characterization factors rather than directly measuring environment-specific factors. (Basili & Rombach 1987).

In general, the project-specific tailoring approach, as a traditional SPI mechanism, adopts a top-down approach. The CMM tailoring guideline, for example, suggests creating a project specific software development plan (SDP) as a result of the project planning activity (Ginsberg & Quinn 1995) at the beginning of the project. The SDP is defined as "a key element in the management of the project" (Ginsberg & Quinn 1995, p. 36) and it should

include the project specific version of the OSSP to be used throughout the project. Thus, tailoring may often be based on the knowledge from the previous projects (Basili & Rombach 1987) and occur once at the outset of the project.

2.3.1.4 Process Deployment

Process deployment can be considered as one instance of SPI in organisations. The deployment may include such activities as piloting the processes, methods, and tools that are identified as potential solutions for the existing goals and problems, and evaluating their effect on the software development. For example, it is suggested that "in the Acting phase of the IDEAL model, solutions to address the areas for improvement discovered during the Diagnosing phase are created, piloted, and deployed throughout the organisation" (McFeeley 1996, p. 4) (Table 1). Thus, the organisational models of SPI, such as IDEAL and QIP, provide an overall framework to be applied in any deployment process. Yet they do not suggest any specific approach for the deployment of new processes, practices and tools. In such a traditional deployment approach, the focus is on the improvement of organisational software processes rather than on deploying practices for the purposes of individual projects.

Basically, an organisation may adopt a big bang or piecemeal approach for deploying new processes, practices and tools. The piecemeal, i.e. evolutionary approach "seems the preferred default approach, implying a prolonged period of growth within the organisation through smaller phased enhancements" (Sweeney & Bustard 1997, p. 266) whereas the big bang, i.e., revolutionary, approach "implies a sudden substantial change to an organisation that results in a distinctly new way of working" (Sweeney & Bustard 1997, p. 266). The IDEAL model, for instance, suggests the piecemeal approach where selected solutions are tested in pilot projects in order to define the needs for tailoring for the rest of the organisation (McFeeley 1996). It is also suggested that one solution may require several pilots and iterations before it has been refined and verified as applicable and ready to be deployed across the organisation (McFeeley 1996). Several studies have demonstrated the appeal of such piecemeal deployment among practitioners (Niazi et al. 2003). However, references can also be found to the preference for big bang approaches in implementing substantial changes rapidly in order to reach the end state more quickly (Jalote 2002). The piecemeal approach has, however, also been criticised for requiring repetitive activation of

different deployment-related mechanisms and also for creating resentment among practitioners due to the constant changes they are requested to implement (Jalote 2002). Furthermore, the big bang approach has been suggested in cases where the business is no longer capable of achieving its purpose, where the external circumstances are changing rapidly or where there is a need to wake up a sleepy organisation in which the motivation and activity of its members need to be improved (Sweeney & Bustard 1997).

2.3.1.5 Measurement

It has been argued that "you cannot control what you cannot measure" (DeMarco 1982). It has also been said that without software measurement the evaluation and improvement of software processes would be impossible (Arthur 1993). Three main purposes of measurement have been identified: understanding development and maintenance, controlling projects, and improving processes and products (Fenton & Pfleeger 1997). Software measurement is defined as a "continuous process of defining, collecting, and analysing data on the software development process and its products in order to understand and control the process and its products, and to supply meaningful information to improve that process and its products" (van Solingen & Berghout 1999, p. 19).

Various measurement mechanisms aim at providing quantitative support for all the key SPI objectives; from understanding the current status to planning improvements and assessing the "rate and level of learning to ensure that gains have in fact been made" (Garvin 1993, p. 79). It has been stated that the software processes as well as their outputs "have measurable attributes which can be observed to describe the quality, quantity, cost, and timelines of the results produced" (Florac et al. 2000, p. 8). Software metrics are used for measuring specific attributes of a software product or a software development process. This is done in order to enhance decision-making by drawing up estimates, tracking the progress, and evaluating the state of quality. The measurements also serve for analysing defects, and validating the best practices for development (Grady 1992). Measurement-based SPI also provides a means for evaluating the effectiveness of the used processes (Pfleeger & Rombach 1994), understanding the effects of implemented improvement actions (van Solingen & Berghout 1999) and, as a result, also for drawing an objective process model (Pfleeger & Rombach 1994).

A wide range of measurement methods provides quantitative mechanisms to support the different activities of continuous SPI in organisations. The GQM (Goal-Question-Metric) method (Basili 1985, 1994, Basili & Weiss 1984), for instance, provides a goal-based approach for defining metrics that, in return, constitute answers to the underlying questions and goals. In GQM, the corporate improvement objectives lay the foundation for improvement initiatives, whereas at the project level the measurement data collection activities are usually performed (van Solingen & Berghout 1999). Statistical Process Control (SPC) (Florac & Carleton 1999, Florac et al. 2000), on the other hand, aims at stable processes with predictable results using statistical software process management.

It has been realised that, often, the software developers do not welcome the effort put into manual metrics collection. In addition, manually-collected metrics data may often be affected by errors, and thus be unusable. Therefore, automated tools such as PROM (PRO Metrics) have been proposed for collecting and analysing metrics data. (Sillitti et al. 2003).

2.3.1.6 Experience, Knowledge and Learning

The value of knowledge, experience and learning should not be underestimated in the process of continuously improving organisational practices. As Deming points out, the "waste of knowledge, in the sense of failure of a company to use knowledge that is there and available for development, is even more deplorable" than the waste of materials, human effort and machine time (Deming 1990, p. 466). Deming also argues that "a company must, for its very existence, make use of the store of knowledge that exists within the company" (1990, p. 466).

Some of the existing SPI methods especially focus on different aspects of knowledge and experience, and on their utilisation within the improvement of organisational practices. Basili argues that quality improvement is often achieved by reusing and modifying a set of elements based on learning from direct experience (Basili & Caldiera 1994). According to Basili, however, learning and reuse of knowledge usually only occur because of individual efforts or by accident (Basili 1989). Furthermore, Basili points out that this inevitably leads to a loss of the experience and knowledge after the project has been completed and suggests a reuse-oriented software development process in which learning and feedback are regarded as integrated components, and experiences

are stored in an experience base called EF (Experience Factory). Rather than considering just knowledge and experience, it is suggested that the EF also includes data and information in the form of metrics, for instance.

Project postmortems, i.e., post-mortem analysis (PMA's) or project retrospectives, have been suggested for harvesting the experience of success and failure in previous projects and are claimed to be a valuable tool for organisational learning (Stålhane et al. 2001) and improving the methods and practices (Collier et al. 1996) for future projects in an organisation. Thus, the project postmortems serve as a traditional feedback mechanism such as in the QIP learning cycle, to provide process knowledge from project teams for organisational improvement purposes. In fact, project postmortems have been claimed to be "an excellent step into continuous knowledge management and improvement activities" (Birk et al. 2002). Different procedures have been suggested for conducting project postmortems (e.g., Collier et al. 1996, Kerth 2001). There are also various techniques available for supporting a post-mortem in a project, such as the time line technique (Kerth 2001), brainstorming (Rawlinson 1981), the KJ method and its affinity diagrams (Scupin 1997), as well as Root Cause Analysis (RCA) using cause and effect diagrams (Ishikawa 1985). Some software development processes, such as the Team Software Process^{SM} (TSP) (Humphrey 2000), have embedded a project postmortem as the final step in the software development activities of a project.

The project-based experiential learning model of Dybå et al. (2004) suggests different kinds of reviews or workshops to be held in a project in order to enhance the learning before, during, and after the project. The initiating workshop should serve the purpose of sharing the experiences relevant to the incipient project among the project members. During the project the project teams may pause to check if the course is right for the project, and to reflect on their experiences in order to conduct short-term improvement actions. The after-project workshops mainly serve the purpose of making the experiences and learning of the project teams available for later projects and organisational learning purposes. (Dybå et al. 2004).

The GQM method also suggests holding feedback sessions to discuss the results of a measurement program (van Solingen & Berghout 1999). Typically, the GQM feedback sessions are arranged by the organisational SPI stakeholders, and

the purpose of these sessions is to utilise the process and context knowledge of the software development teams to interpret the metrics data collected from the project. The collected feedback is used to support both the organisational SPI and future projects in the organisation.

3. SPI in Agile Software Development

In this section there is a discussion about how the traditional elements of SPI have been addressed in the context of agile software development. In addition, the differences between SPI in the contexts of traditional and agile software development are summarised and the resulting implications are discussed.

3.1 The Elements of SPI in Agile Software Development

When considering the relationship between agile software development and SPI, there are three principles, in particular, of the agile manifesto that deserve attention: the valuing of individuals and interactions over processes and tools, the principle that encourages regular reflection by software development teams in order to become more effective, and the self-organisation of software development teams. Taking regular improvement within project teams as one of the twelve principles of agile software development highlights the importance of continuous improvement also in the agile software development context. In order to welcome changes throughout the agile software development project, whether they concern product requirements or technical aspects, the software process with its practices, methods, and tools must be able to adapt to the specific context while also to respond to the changes when needed.

In the following sub-sections, the current methods, research and discussion on the different SPI elements, in the context of agile software development, are reviewed.

3.1.1.1 Organisational Models

A limited amount of references were found to directly address the issue of organisational SPI within the context of agile software development. One reason for this might be that the focus of numerous agile methodologies is on the project level activities of software development.

Discussion has arisen about how compatible the fundamentals and methodologies of agile software development are with standard process models

(e.g., Paulk 2001) based on the organisational models of SPI, e.g., IDEAL℠ (McFeeley 1996). Significant organisational tensions have been detected "as the stability that underpins notions of quality control is overlaid on environments in flux" (Lycett et al. 2003, p. 79) due to the differences between standardised engineering approaches and agile methods. In addition, Boehm and Turner (Boehm & Turner 2005) have addressed the challenges of management in implementing agile processes in traditional development organisations; the difficulty of scaling up and integrating agile methodologies into traditional, top-down systems of development organisations has been identified as a major challenge. Nonetheless, it is equally important for both approaches to pursue the aim of "quality of product, service, and process to gain market presence and competitive edge" (Lycett et al. 2003, p. 79).

The difficulty of adopting agile methods in, for instance, the CMMI® based environment and the lack of guidance on how to take advantage of the existing best practices of an organisation in transition towards agile methods, has been identified as one of the major obstacles in the co-existence of the agile and traditional approaches (Reifer 2003). In the study of Kähkönen (2005), the SW-CMM based IDEAL model is considered incompatible as an SPI reference model for use in the agile software development context. The central limitations identified in the IDEAL model were the organisational focus of the model, as opposed to the strong project focus of the first agile development initiatives, and the conflict of conducting SPI in a plan-driven manner while the adopted method itself was agile. As a result, Kähkönen proposes a light life-cycle model especially for projects deploying agile methods.

While the problems of integrating agile and traditional software development at the organisational level have been identified, there now seems to be an urgent need to explore their potential solutions.

3.1.1.2 Standard Processes and Assessments

The compatibility of agile software development approaches with the existing standard process models is one SPI issue that has been addressed in agile literature. One central problem has been posed as follows: "How do you merge agile, lightweight processes with standard industrial processes without either

killing agility or undermining the years you've spent defining and refining your systems and software engineering process assets?" (Boehm & Turner 2005, p. 30).

Some agile proponents have argued that "people willing to spend money on CMM® certification are less interested in the agile value proposition, while those needing agility for business reasons are less interested in getting CMM or ISO 9000 certification" (Williams & Cockburn 2003, p. 40). Nevertheless, mature software organisations especially are concerned about how the adoption of agile processes will affect their assessment ratings (Boehm & Turner 2005).

It has been argued that the synergy (Paulk 2001) and philosophical compatibility (Reifer 2003) of XP and CMM® have been agreed upon among most of the leaders in the field (Reifer 2003). However, shortages also between the ISO and CMM requirements and agile methodologies, such as XP or Scrum, have been reported, along with a lack of practices to support the commitment of management to the defined software development process, and also regarding the setting up and staffing of an independent quality assurance group (Vriens 2003). In addition, the degree of documentation and the infrastructure required by current process standards for lower-level certification are issues of concern (Boehm & Turner 2005).

An urgent need has been recognised for a set of guidelines for agility-compatible standard process maturity assessments and also for a set of standards for the acknowledgement of agile methods by lead assessors (Boehm & Turner 2005). In linking the agile methodologies and quality standards, such as ISO 9000 and CMMI®, Lycett et al. (2003) have proposed a framework for mapping the candidate process pattern elements, for example from Rational Unified Process (RUP) (Kruchten 2000) with CMMI® to "supplant repeatability with consistency, while still providing the audit trail necessary for assessment" (Lycett et al. 2003, p. 84). In addition, experience reports have been published suggesting that a certain level of certification of, for example, XP based process is possible (Paulk 2001), although it has also been suggested to require adoption of additional software development practices (Kähkönen & Abrahamsson 2004, Vriens 2003).

The agile assessment method has been suggested as providing a lightweight approach for assessment to identify and adopt the most suitable agile methods

amongst the existing organisational practices (Pikkarainen & Passoja 2005). Furthermore, techniques have been suggested for increasing the agility level of a software development team by assessing the current agility level against the defined agility goals (Lappo & Andrew 2004). Thus, the current discussion of process assessments in the agile context does not so much address the certification or define the maturity of the organisational software development processes, but rather evaluates the purpose of adopting agile practices.

From 2003 onwards, however, the IEEE Standards Association has conducted agile standardization work in the IEEE 1648 working group (http://standards.ieee.org/board/nes/projects/1648.pdf) to establish and manage software development efforts using agile methods. In the beginning of 2007, the "P1648 – Recommended Practice for the Customer-Supplier Relationship in Agile Software Development projects" is in its draft format.

3.1.1.3 Process Tailoring

In agile software development, the concept of universal and repeatable processes is considered defective and it has been proposed instead that "each situation calls for a different methodology" (Cockburn 2002, p. 84). Thus, agile software development calls for flexible software development processes that have been defined as "…less precise than rigorous ones. For example, the development processes may not be defined formally, they utilise guides rather than rigid rules, or they may be applied differently by different teams in the same company or even within the same product group. A flexible process may not produce the *same* results every time, but the results are *similar* enough for a written description of the process to benefit the organisation" (Highsmith 2000, p. 228).

Agile specific methods are needed to tailor agile software development practices within individual projects and within the entire organisation. References can be found in the agile software development literature to the tailoring activity at the beginning of the project (i.e., static tailoring) as well as to the continuous process adaptation, which takes place throughout the life-cycle of an individual project (i.e., dynamic tailoring).

Keenan (2004) suggests three strategies for process tailoring in the context of agile software development: 1) using a comprehensive pattern based process framework, such as RUP, as a pool for selecting appropriate elements at the beginning of each project, 2) defining a set of processes, as in Crystal, and selecting the best match for the project at hand with possible fine tuning, and 3) defining a tailored process for the project by blending ideas and techniques from best practices and local experience. The two former ones are considered as static tailoring prior to the project, as is characteristic of traditional software development, whereas the last strategy is suggested as a dynamic approach for tailoring occurring also during the development (Keenan 2004).

Lycett et al. (2003) have proposed that one primary challenge between agile principles and plan-driven software development is the balancing between implementing repeatable processes while still allowing the nuances in a particular development context. They also discuss the gap between the management expecting predictable processes to produce the highest quality with minimal cost and factual software development where uncertainty and contextual differences are present (Lycett et al. 2003). As a result, a more agile approach to tailoring was proposed by: 1) factoring a core set of process artefacts suitable for a wide range of developments, 2) identifying candidate patterns for core types of development as well as for contextual application of activities, artefacts, and guidelines, and 3) producing a skeletal framework for selecting patterns based on project, product, team, and organisational characteristics (Lycett et al. 2003). This pattern-based approach for agile process tailoring also suggests that the reflection of project teams on the experiences of applying selected patterns should contribute to the organisational pattern catalogue. However, the framework does not suggest any specific procedures for attaining the organisational improvement of patterns.

Cockburn has suggested a methodology shaping technique for tailoring a starter methodology for a project and building an organisational library of experiences (2005). This methodology consists of interviews to harvest the experiences of the project team members and members of other projects, as well as a brainstorming workshop where the starter methodology is agreed upon. The methodology is based on the fixed rules of software development in the organisation, while it also relies on the “liked/keep” and “disliked/avoid” decisions of the project team. Thus, the formulation of the software development

process is strongly grounded on the experience and knowledge of the software developers.

In APM (Agile Project Management), a specific practice of "process and practice tailoring" is introduced. Its objective is to define "the approach the project team will use to deliver a product" (Highsmith 2004, p. 118). The suggested tailoring approach "starts from the organisation's standard framework, and then the project manager and team tailor it to their needs within the framework's constraints" (Highsmith 2004, p. 118). In addition, it is stated that nothing in APM is static – neither the product nor the practices – and the teams are "encouraged to adapt everything except the base essential policies and process framework to the reality of the actual situation as it unfolds" (Highsmith 2004, p. 119). Thus, the APM process and practice tailoring approach includes both static tailoring at the beginning of the project and dynamic tailoring throughout the project, to be conducted even iteratively. Furthermore, APM recognises the broad process framework set by the organisation, which can be adapted by the team with certain limitations.

The dynamic process tailoring, especially during and within the ongoing software development projects, has been highly valued in the principles of agile software development. The agile principle of "regular team reflections of software developers in order to become more effective" relates directly to the continuous and dynamic project-specific tailoring activity, whereby the organisational base process is iteratively tailored throughout the project by the software development team. Furthermore, the self-organising principle of agile software development dictates that "the working framework should grant the team as much flexibility and authority to make decisions as possible" (Highsmith 2004, p. 220). Thus, numerous references to process adaptation can be found in the existing agile methodologies. Interestingly enough, the first versions of XP (Beck 2000) and Scrum (Schwaber & Beedle 2002) still largely ignored the aspect of tailoring, team reflections and SPI. In the later versions of these methods (Beck & Andres 2004, Schwaber 2004), however, proposals for conducting such practices have been made (Table 3). The use of retrospectives throughout the project has been suggested to solve the problem of traditional retrospectives, as the changes can be immediately incorporated into the project's processes (Koch 2005). The problem of traditional methods has been the lack of mechanisms to ensure subsequent utilisation of the lessons learned from finished

projects (Koch 2005). Furthermore, Koch argues that, while there still is the problem of sharing the learning beyond single software development teams, the shift of personnel from one project to another enables the memorisation and transfer of practical solutions.

Based on a number of reported SPI initiatives, Komi-Sirviö (2004) has built a framework of Critical Success Factor (CSF) criteria to be used to evaluate SPI methods. In the framework, a total of 15 factors found to facilitate successful SPI have been identified. In Table 2, the CSF factors identified by Komi-Sirviö are presented. The columns titled "Main class", "Sub-classes", "CSF", and "Evaluation" present the definitions of the original framework. In addition, the column of "Evaluation of agile process adaptation" suggests an interpretation of each CSF in the context of any successful SPI method claimed to be suited for process adaptation among agile software development teams.

Table 2. CSF Framework and its Interpretation among Process Adaptation Methods of Agile Development Teams.

Main class	Sub-classes	CSF	Evaluation	Evaluation of agile process adaptation
Improvement Management	General guidance	1	Does the method support different SPI approaches?	Is the method linked to an overall improvement approach? Are alternative techniques proposed for use in the process adaptation? Are organisational guidelines suggested to support project specific process adaptation?
	Staffing the SPI initiative	2	Does the method support the participation of all affected parties?	Are all the parties identified and involved in those practices that on which the process adaptation has an effect?
		3	Does the method support co-operation with software engineers?	Vice versa: Does the method support team co-operation with other organisational SPI stakeholders?
	Training	4	Does the method support planning and carrying out training as a part of the initiative?	Is there an organisational facilitation available to assist software developers in practice?
Commitment	Manager Commitment	5	Does the method support the commitment of top managers?	Are the top managers provided with feedback on the SPI activities of process adaptation among project teams (why, how, how effectively was the process adapted)?
		6	Does the method support the commitment of middle managers?	Are the middle managers provided with feedback on the SPI activities of process adaptation among project teams (why, how, how effectively was the process adapted)?
	Engineer Commitment	7	Does the method support the commitment of software engineers	Is the software development team provided with support to implement the process adaptations?
Culture		8	Does the method support developing improved solutions on a case-to-case basis?	Does the method support the process adaptation based on the context specific needs and daily problems of the software development team?

Main class	Sub-classes	CSF	Evaluation	Evaluation of agile process adaptation
Plan	Current State Analysis	9	Does the method support clarifying the current status of processes?	Does the method support identifying the strengths and weaknesses of the daily working practices of the software development team?
	Goal Definition	10	Does the method support establish a link between the business and improvement goals?	Does the method support continuous co-operation between organisational SPI stakeholders and agile software development teams?
		11	Does the method support measurable improvement goals?	Does the method support the setting of goals for process adaptation actions?
	Improvement Planning	12	Does the method support generating an improvement plan?	Does the method provide a means for generating an improvement plan?
Do		13	Does the method support the testing of developed solutions in a pilot project?	Does the method support the follow-up of the process adaptation actions? Does the method support the validation (qualitative/quantitative) of the process adaptation actions during the ongoing project?
Check		14	Does the method support using metrics in monitoring improvement actions and results?	Does the method support the organisational monitoring of process adaptation actions?
Act		15	Does the method support the sustainability of an improvement initiative?	Does the method support the storing of the process adaptation knowledge of software development teams? Does the method support organisational utilization of process adaptation knowledge of software development teams?

Table 3 presents how the current agile software development methods of dynamic tailoring seem to meet the previously defined criteria of the CSF framework (Table 2). In the evaluation, the following marks are used to define

how each method/activity meets the CSF: 1) X = the means to accomplish the issue has been defined, 2) / = the importance of the issue has been acknowledged, and 3) - = the issue has not been included or considered in the method/activity.

Table 3. Evaluating the SPI Activities of Agile Process Adaptation.

Method of Origin	**Process Adaptation Activity/Technique**	**CSF**														
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
XP	Reflection (XP principle) (Beck & Andres 2004)	-	/	-	-	-	-	-	/	/	-	/	-	-	/	-
Scrum	Sprint Retrospective Meeting (Schwaber 2004), (Schwaber & Beedle 2002)	-	X	/	X	-	-	-	X	/	-	-	-	-	-	-
Crystal	Reflection workshop technique (Cockburn 2002, 2005)	-	X	-	-	-	-	-	X	X	-	-	X	-	-	-
APM	Process and practice tailoring (Highsmith 2004)	X	X	-	-	-	-	-	X	X	-	-	-	-	-	-
-	Postmortem review technique (Dingsøyr & Hanssen 2002)	-	X	-	-	-	-	-	X	X	-	-	-	-	-	/

As can be seen in Table 3, the existing agile mechanisms of process adaptation seem to fall short regarding the aspect of general guidance (CSF 1). Only APM seems to address the idea of the project teams actually working within an organisational framework of processes and practices, and propose that organisational guidelines should exist to support the process adaptation of software development teams (Highsmith 2004). Furthermore, some of the methods suggest specific techniques to be used in SPI but do not seem to offer any alternatives.

All the suggested techniques seem to address the staffing of the SPI initiative to some degree. Each of the methods identifies the stakeholders who should attend the process adaptation within agile software development projects (CSF 2). Naturally, the central group of actors identified by all the adaptation approaches is the software development team itself. However, Scrum (Schwaber 2004) also

promotes the role of the Scrum Master in facilitating the process adaptation activities as well as acting as a middleman to gain organisational support for the problems of the project team (Schwaber & Beedle 2002). In other words, in Scrum the importance of collaboration between the organisational and project level stakeholders in SPI has been addressed (CSF 3). In addition, the proposed facilitator in Scrum also relates to CSF 4 (training), as in the CSF framework this has been defined as one way of providing training for software development teams in their SPI initiatives (CSF 4).

Neither the role of top (CSF 5) and middle managers (CSF 6) nor their commitment has been addressed in the existing process adaptation techniques. None of the methods, for example, suggests that the management should be provided with knowledge and feedback on how the individual project teams are adapting their processes and how successfully. Nor has there been a discussion in the existing methods about how the commitment of the software developers (CSF 7) should be addressed to ensure their regular effectiveness in conducting process adaptation. An important aspect for such commitment, especially in the agile software development context, might be the constant support and feedback of the organisational SPI stakeholders for the developers.

Commonly, the agile specific methods of process adaptation propose that the adaptation should be conducted within individual project teams and that it should be based on the experiences of success and failure of software developers. Thus, the context specific adaptation and the case-to-case basis for identifying improvements (CSF 8) are in a central role in the methods. This also refers to the fact that the methods suggest that the adaptation should be, at least to some degree, based on an iterative identification of the current status of the software development process of the software development team regarding its weaknesses and rewards (CSF 9). Thus, rather than defining the current status of the organisational software development process, agile process adaptation focuses on evaluating a specific instance of the process in a specific instant of time in a specific software development project. In addition, the Crystal, APM, and postmortem review techniques suggest specific techniques for assessing the current problems in the process. Crystal, for instance, suggests using reflection workshop flip-charts, APM advocates using a team self-assessment chart, and the postmortem review technique proposes the KJ method of Scupin (Scupin 1997).

The existing process adaptation methods do not specifically discuss the definition of goals (CSF 10 and 11) in the process adaptation. In project-centred process adaptation, the link between the business goals with the project level SPI actions (CSF 10) may not be of immediate concern from the viewpoint of an individual project team. However, an organisation may still wish to establish the linkage between the two levels. In this case, the SPI co-operation between the two levels should be addressed. In any case, every SPI action of the software development team should be clearly linked to a specific improvement goal – whether organisation or project-specific – to provide a basis for appropriately evaluating whether the improvement has been successful (CSF 11, CSF 13, and CSF 14). Furthermore, the current process adaptation methods seem to focus on the identification of *causes* of improvements rather than the goals. For example, the postmortem review technique suggests a root-cause analysis for clearly determining the cause of problems.

Surprisingly, the existing methods largely lack the means of adequately defining and planning (CSF 12) the process adaptation as well as means for systematic follow-up and validation (CSF 13) of the changes that the team implements in the process. Traditionally, however, these have been considered central aspects of SPI. The reflection workshop method (Cockburn 2005) includes a generation of flap-sheet "keep these" and "try these" issues to be posted on the wall in the office-space of software developers. The suggested output format, however, still lacks the aspects of "what exactly", "how", "who", "when", and "why", for instance. As a result, there is no means for the project team to recall exactly what is needed to be done nor to assess whether the project team actually followed the process adaptations as agreed (CSF 13), or if the changes were successful or not (CSF 13). In fact, the validation of the SPI actions taken by the project team – quantitative or qualitative – has not been addressed in the existing methods in any way. In XP it is still acknowledged that the project team should be able to evaluate the effects of the changes if the team is allowed to adapt their daily working procedures (Beck 1999). However, the iterations of agile software development seem to provide an ideal context for conducting process adaptations in a systematic and validated manner in which the SPI actions are implemented and tested rapidly in an iterative fashion during the ongoing project.

An organisational monitoring of project level process adaptation activities (CSF 14) or an organisational utilisation of project specific SPI knowledge for organisational SPI purposes (CSF 15), both seem to be out of the scope of the existing agile software development methodologies and their process adaptation activities. In summary, it could be argued that the existing process adaptation methods currently provide very little information on how to actually conduct process adaptation. Some of the methods suggest specific techniques to be used in some specific tasks. At best, techniques are proposed for identifying the problems and their causes in the process, and for discussing improvement actions. However, extensive procedures for conducting process adaptation activities systematically within agile project teams do not seem to exist and many aspects identified as critical SPI success factors are not addressed, including the systematic and validatory nature of SPI actions and the organisational involvement in the project level SPI activities.

Many of the agile methodologies also refer to the traditional PMA techniques for conducting process adaptation (e.g., Cockburn 2005). Some of the tasks of process adaptation may, indeed, largely benefit from the techniques of traditional PMAs. However, the PMAs also lack mechanisms for piloting, validating and following up improvement actions, for instance, as they are simply designed to harvest knowledge from finished projects instead of improving the practices of ongoing projects.

3.1.1.4 Process Deployment

It has been stated that the way of introducing agile processes in an organisation has a significant impact on the ultimate success of the process change (Cohn & Ford 2003). It has been also suggested that agile processes should be introduced one technique at a time and address the most pressing problem first (Eckstein 2004). Many of the current agile methodologies seem to support the “big bang” approach or at least they do not seem to provide any criteria for a stepwise adoption of the suggested practices, methods and tools. For example, Crystal, in its early version (Cockburn 2005), simply proposes that the size of the project and the criticality of the product should be used as the selection criteria between the lighter and heavier versions of the Crystal family of methodologies.

RUP (Kruchten 2000) provides a large number of process artefacts that can be selected for a process model. RUP proposes a step-by-step process for its implementation in an organisation (Kruchten 1999). The deployment process consists of six steps: 1) assess the current state, 2) set (or revise) goals, 3) identify risks, 4) plan process implementation, 5) execute process implementation, and 6) evaluate process implementation. Thus, the process seems to be well in line with the organisational models of SPI (2.3.1.1) and it also provides detailed guidelines for deployment. In RUP, it is also proposed that a pilot project should be conducted before extending the RUP practices to the entire organisation (Kruchten 1999).

One major obstacle has been identified in adopting agile methodologies: the lack of mechanisms to adopt agile methods within projects with established plan-driven processes (Reifer 2003). The mixing of plan-driven and agile approaches is also considered acceptable as it can benefit from the strengths while also compensating for the weaknesses of these approaches (Boehm & Turner 2003b). It has also been suggested that a gradual transition from plan-driven into agile processes could make the change easier for the development team (Cohn & Ford 2003). Cohn and Ford (2003) have referred to a number of possible setbacks or errors that may occur when an organisation is transitioning from plan-driven towards agile processes. The problems are mainly caused by resistance to, or overenthusiasm for, agile practices within a software development team. Many of the problems also seem to be caused by the lack of transparency between the different organisational levels resulting from undefined and inadequate operations between agile project teams and different organisational stakeholders. Such problems may be due to, for example, a lack of established organisational practices for project-tracking, an inadequate understanding of the payback of new software development practices such as pair programming, and a lack of co-operative mechanisms in the project teams and at the organisational level. Further factors which can cause problems are an ill-defined means of collaboration between the agile project and other development teams, and the effect of agile methodologies on customer contracts regarding such issues as product features, project duration and cost (Cohn & Ford 2003).

Kähkönen (2005) has proposed a life-cycle model consisting of high-level steps to support improvement projects deploying agile practices. The deployment steps identified in the model include decision-making on behalf of agile

methods, selecting an approach for deployment, selecting methods to deploy, initial deployment, iterative adaptation of process, and ensuring the use of process knowledge in the future. When addressing the project level improvement activities, the model of Kähkönen also indicates that the best way to retain and make use of the accumulated process knowledge is to move the whole team to a new assignment or to organise project post-mortems like those used in plan-driven development models. The model does not, however, address the organisational adoption of agile practices or suggest any specific methods to be used in the deployment process. Despite the fact that agile assessment (Pikkarainen & Passoja 2005) aims at providing a means for identifying and selecting agile practices in a goal driven and context specific manner (see also 3.1.1.2), the organisational methods identifying and selecting potential agile methodologies have been sparse.

3.1.1.5 Measurement

It has been stated that the traditional "measurement techniques might be inadequate to support agile processes' rapid pace" (Boehm & Turner 2005, p. 34). One reason for this can be found in the highly different work breakdown structures of the two approaches. In agile software development, the traditional progress measures, for instance, have been successfully substituted with backlog lists of stories (Schwaber & Beedle 2002) and their state of completion (Boehm & Turner 2005).

The previous claim applies not only when considering the measurement as a controlling tool in projects, but as a tool for improving processes and products in agile software development. On the one hand, organisations conducting agile projects are still likely to need measurement as traditionally used in organisational SPI (2.3.1.5), but also in supporting the more agile-specific activities of software development teams. As discussed in the earlier sub-sections, the current SPI activities in agile software development methodologies, however, largely ignore the issue of metrics and validation. In addition, the project level SPI activities are still not, to a large extent, linked to organisational SPI activities where, traditionally, metrics have played a significant role. It seems, in fact, that measurement is not much addressed among the different SPI related issues within the context of agile software development. In certain agile

software development methodologies, however, some references can be found to the topic. In XP, for instance, it has been acknowledged that the process adaptation within software development teams requires mechanisms for assessing the effects of the changes (Beck 1999). Traditionally, metrics have been the main tool of such evaluation. Highsmith (2000), on the other hand, has discussed the uncertainty of the problem-solving being always able to find the right solution. He has argued that "while we can surround problem-solving with more rigorous data-gathering or documentation processes, it is important that we remember that problem-solving is an emergent process – one that defies strict cause-and-effect analysis" (Highsmith 2000).

In agile literature, however, the importance of metrics has been addressed in, for example, project management. For instance, in the Scrum approach for project management it is suggested that "if you measure the right things, improvements can be made" (Schwaber 2004, p. 114).

3.1.1.6 Experience, Knowledge and Learning

Experience, knowledge and learning are highly valued in the context of agile software development, especially among individual software development teams. One clear example of the emphasis on collective learning and experience of software developers, is the agile principle suggesting that the teams should regularly reflect on their work in order to become more effective and to be able to tune their behaviour accordingly. In XP, the role of team learning is defined as: "good teams don't just do their work, they think about *how* they are working and *why* they are working. They analyze why they succeeded or failed. They don't try to hide their mistakes, but expose them and learn from them" (Beck & Andres 2004, p. 29). From another angle, learning among agile software development teams is based on the experience on "doing". In other words, "learning is action reflected" (Beck & Andres 2004, p. 30). It has even been claimed that "the Agile methods all recognise the importance of the learning that both the customer and developers experience" (Koch 2005).

In agile software development, the face-to-face communication is emphasised also in learning activities. For example, in the agile process adaptation activities of different agile software development processes the idea is for the project team to regularly gather together to discuss and collectively learn in order to improve.

The role of facilitator has also been highlighted, especially in holding effective face-to-face meetings (Highsmith 2000). In addition, traditional postmortems have been suggested to be conducted after agile software development projects, e.g. in ASD (Highsmith 2000).

3.2 Comparison of SPI Elements in Plan-Driven and Agile Software Development

In this section, the earlier discussion of software development and different elements of SPI in the context of plan-driven (2.3.1) and agile (3) software development approaches are summarised. The aim of this section is to compare the fundamentals of traditional and agile software development from the viewpoint of the six identified SPI elements (i.e., organisational models, standard processes and assessments, process tailoring, process deployment, measurement, as well as experience, knowledge and learning) and to discuss how the different viewpoints of SPI are relevant in this line of research.

As Table 4 illustrates, some of the fundamental aspects of software development in plan-driven and agile approaches can be considered contradictory. For one, the traditional goal of a software process is to provide high predictability, stability, and repeatability using highly managed and quantitatively monitored software development processes. On the other hand, agile principles highlight the need for the software process to be flexible, to be able to rapidly respond to the constant changes and context specific needs of software development. As a result, traditional software development emphasises up-front contract negotiations where the requirements, cost and schedule of the product development are fixed (i.e., schedule) and the end product will be delivered at the end of the project lifecycle. In this mode of software development, traditionally, extensive documentation and quantitative monitoring of the product development process plays a central role. The principles and practices of agile software development, in turn, address the constant changes. For instance, it is suggested that the requirements may not always be definable up-front, and that higher customer satisfaction and a higher quality of end products can be accomplished through continuous collaboration with the customer and incremental delivery of working software. In this mode of development, the

emphasis is put on face-to-face communication, self-organisation of teams, and flexible and context specific software development processes.

Table 4. Fundamentals.

Characteristic	Plan-Driven View	Agile View
Software Process	High predictability and stability of software development (Lycett et al. 2003)	Rapid response to constant changes
	Repeatable, well managed, and measurable software processes	Context specific and adaptable software process
Software Development	Goals of product and productivity fixed beforehand	Product requirements rapidly change throughout the development
	Extensive documentation	Light documentation (simplicity = the art of maximising the work not done)
	Quantitative monitoring	Face-to-face communication
	One-off delivery of end-product	Continuous, iterative delivery of working software
	Monitoring and management of software development	Self-organisation of teams Regular team reflections to become more effective

From the viewpoint of the different SPI elements, the differences between the traditional and agile approaches are, to some extent, contradictory. In Table 5 the traditional viewpoint of organisational SPI is set against the agile viewpoint.

Table 5. Organisational SPI Models.

	Characteristic	Plan-Driven View	Agile View
Organisational SPI Models	SPI Approach	Top-down	Bottom-up
	Primary Goal	Organisational procedures for improving the organisational software process(es)	Adapting the process to the contextual needs of individual project teams
			Improving the effectiveness of individual project teams
	SPI Control	Organisational control of SPI	Self-organisation of teams
	Knowledge Transfer	Explicit knowledge: external knowledge capture and inert knowledge transfer to support a learning paradigm (Lycett et al. 2003)	Face-to-face communication Tacit knowledge: Establishing and updating project knowledge in the participants' heads rather than in documents. (Boehm & Turner 2005)
	Basis for Improvement	Organisational Goals Measurements	Contextual needs Experience and Learning of Software Developers Regular team reflections

Traditionally, SPI has been approached in a top-down manner, in which the organisational level has played a major role in defining the goals of SPI and planning, managing, and controlling the SPI initiatives. In the agile software development context, on the contrary, the experience and knowledge of software developers and the self-organisation of software developers in improving and adapting their daily working practices have been clearly placed in a central role. In the agile approach, the role of management is to organise and co-ordinate rather than plan, execute, or control. Furthermore, the contextual needs for improving and adapting processes throughout the entire development process are emphasised, rather than the organisational goals in the regular SPI activities of development teams.

In Table 6, the process standards and assessments are compared in the context of traditional and agile software development.

Table 6. Standard Processes and Assessments.

Standard Processes and Assessments	Characteristic	Plan-Driven View	Agile View
	Primary Goals(s)	Assess and improve the organisational software process against a standard reference process model	Assess and improve the organisational software process by identifying the potential agile practices
		Evaluate the maturity of the organisation as a whole Evaluate the capability of individual process areas of the organisation	The standard based evaluation of maturity not currently addressed in agile sotware development[1]
		Certify the maturity level of the organisation	

Traditionally, standard process models, such as CMMI, have played a central role in organisational SPI initiatives (Table 6). Different assessment methods have been used to provide support for evaluating the maturity and capability of organisational software processes against the standard process models. In agile software development, the standardization work is currently ongoing. This work aims at the recognition of agile methodologies, practices and principles as part of commonly accepted standard process models. Currently, the assessment methods in standard processes do not define how to appraise agile software development processes. Various independent assessment methods have, however, been suggested to provide organisational mechanisms for the assessment and selection of potential agile methods.

In Table 7, the process tailoring aspect of SPI is considered from the viewpoints of traditional and agile software development.

[1] Currently, the standardization work of the IEEE 1648 working group (http://standards.ieee.org/board/nes/projects/1648.pdf) for establishing and managing software development efforts using agile methods is ongoing. The relation of agile software development and standard processes and assessments is yet undefined.

Table 7. Process Tailoring.

	Characteristic	Plan-Driven View	Agile View
Process Tailoring	Primary Goal(s)	Continuous tailoring/improvement of organisation's standard software process(es) (OSSP)	Tailoring (at the beginning of a project) a project specific software process from OSSP using organisational tailoring guidelines
		Static tailoring, i.e., one-off, up-front definition of a project specific software process using organisational guidelines and process libraries)	Dynamic tailoring, i.e., continuous adaptation of project specific software processes during the ongoing project Self-organisation of development teams
	Supporting Mechanisms	Organisational goals, SPI initiatives, Pilot projects, Process Measurements, Process Assessments, Project post-mortem, observation, brainstorming sessions, surveys, analysis methods, etc.	Comprehensive pattern based process framework Defined set of organisational processes to select the best method for the project
		Organisational tailoring guidelines Organisational process library Quantitative selection of project specific methods and tools Databases and process maps to capture criteria and information for process tailoring	Regular team reflections in a face-to-face manner for explicit collective learning and process adaptation Organisational guidelines for tailoring

In the fundamentals of agile software development, the process tailoring is likely to be the most visible and central SPI element. Traditionally, process tailoring is often understood as an activity in which the organisational software processes are modified or where the organisational process is tailored for a specific project as a one-off activity at the beginning of a project (Table 7). This activity has been traditionally supported by organisational process libraries, standard process models, tailoring guidelines, and quantitative mechanisms for evaluating suitable methods and tools. In addition, the role of capturing, storing and analysing learning from previous projects (e.g., project postmortems) has been traditionally a main means of defining the existing practices of software development and organisational guidelines for process tailoring. In the agile context, on the other hand, the tailoring has been characterised as on ongoing dynamic process in which the experience and collective face-to-face learning of

the software development team is the main source for guiding the context specific process adaptation. Thus, unlike in traditional SPI, the process adaptation in agile software development can be considered to apply a bottom-up approach in contrast to the more top-down project level process tailoring. So far, this activity has been only loosely linked with organisational SPI. Furthermore, only rarely (e.g., Highsmith 2004) has the relevance of organisational framework of processes and practices, or that of organisational guidelines, in the process adaptation of agile software development teams been acknowledged.

In Table 8, the process deployment mechanisms of traditional and agile software development are compared.

Table 8. Process Deployment.

	Characteristic	**Plan-Driven View**	**Agile View**
Process Deployment	Primary Goal(s)	Introducing new process model/ methods/tools in an organisation	Introducing agile processes in projects Transitioning from traditional to agile processes
	Supporting Mechanisms	Organisational SPI Models Piloting	Piloting Iterative deployment
	Approach(es)	Big-bang/Piecemeal	Big-bang/Piecemeal

An organisation can select between the big-bang and piecemeal approaches in the deployment of agile or traditional software development process models, methods and tools. In traditional deployment, the organisational viewpoint of piloting, and evaluating new practices and their deployment in organisational software processes have been in a central role, which has been supported by organisational SPI models. The agile deployment mechanisms have largely focused on mechanisms for supporting the deployment of new practices especially within individual project teams. For example, the iterative process model has been advocated for a piece-meal deployment of agile practices (Kähkönen 2005). In addition, some specific methods have also been recently suggested for supporting the identification of potential agile practices, methods and tools in organisations (e.g., Pikkarainen & Passoja 2005). Agile software

development has been said to lack mechanisms to support the deployment of agile practices in organisations which currently often base on maturity oriented plan-driven processes (e.g., Boehm & Turner 2005).

Table 9. Measurement in SPI.

	Characteristic	Plan-Driven View	Agile View
Measurement	Primary Goal(s)	Identify weaknesses in software process and set improvement goals Identify improvement opportunities in software process Verify the effect of process improvements Monitor the process improvement initiatives	Managing of software development projects

In SPI, traditionally, process measurement (Table 9) plays a central role. For example, metrics have been used to identify weaknesses in the current status of organisational software development, setting improvement goals, verify the effects of improvements and also to monitor SPI initiatives by organisational SPI stakeholders. In the context of current agile SPI mechanisms, however, the role of measurement has not been specifically addressed. The need to evaluate the effects of the process changes made by the software development teams (e.g. through regular team reflections) has been addressed (Beck 1999) but no quantitative mechanisms have been suggested. The role of metrics has, however, been addressed more in the context of agile project management (Highsmith 2004, Schwaber 2004).

Table 10. Experience, Knowledge, and Learning.

	Characteristic	Plan-Driven View	Agile View
Experience, Knowledge and Learning	Primary Goal(s)	Harvesting and utilization of knowledge and experience to improve organisational practices Supporting SPI initiatives	Improving and adapting the software process continuously during the ongoing project
	Supporting Mechanisms	Storing the experience in a database for retrieval & analysis Project PMA's Interpreting metrics data	Regular team reflections to identify weaknesses and improvement opportunities in software process

While measurements play a central role in traditional SPI, the role of experience, knowledge and learning has been highly valued in agile software development (Table 10). Traditionally, the knowledge and learning of software developers has been utilised in SPI by harvesting experiences from finished projects for organisational analysis and for improving organisational software development practices for future projects. The contextual knowledge of software developers has traditionally also been utilised to interpret metrics data collected from pilot projects, for instance.

The role of experience and learning is somewhat different in agile software development. The benefits brought by learning to immediately improve the practices of the ongoing project are emphasised by conducting regular team reflections among the software development teams. Thus, the experiences and learning of software developers are used to guide the SPI within the project teams. For example, the software developers make decisions on how they will change their daily working practices by analysing, e.g., the impediments of the previous iteration. Thus, whereas in traditional SPI the metrics play an important role in the identification of SPI goals, the knowledge and learning of software developers is the corresponding tool in agile context.

Currently, organisations are often running traditional and agile software development projects simultaneously. Thus, it has been realised that there is a need to fit agile software development methodologies and ideologies into organisations using mainly established and mature plan-driven processes (e.g., Boehm & Turner 2005) and to extend the scope of existing agile software development methodologies. Thus, there is a strong case for balancing the currently dominating traditional SPI ideologies and methodologies with the fundamentals of agile software development in order to benefit from the strengths of these approaches while also compensating for their weaknesses (Boehm & Turner 2003b, Lycett et al. 2003).

Despite some fundamental differences, both the traditional and agile approaches to SPI include valuable aspects that should be taken into consideration when contemplating SPI in organisations. From the first viewpoint of this research, i.e., project level SPI of agile software development teams, there are certain issues of traditional SPI that cannot be overlooked. For one, even though the team reflections of agile software development teams place high emphasis on the

learning and process (improvement) knowledge of software developers, the measurement of the software process and product to support quantitative and qualitative verification of the process improvements might still be in place. In addition, especially in large organisations, it may not be appropriate to leave the individual project teams to self-organise and adapt their process without any management control. This highlights the importance of defining the organisational guidelines for the dynamic process tailoring that takes place within teams throughout the software development project.

From the second focus of this research – i.e., integrating the SPI activities of agile software development teams and continuous organisational improvement – there are certain issues of agile software development that cannot be overlooked. Firstly, it seems as if the face-to-face communication and collective and iterative learning among agile software development teams provides new kinds of opportunities for organisations to learn. However, the traditional mechanisms for harvesting, storing, analysing and disseminating process knowledge from projects to the organisation might require significant adaptation to fit the agile software development context. For example, the light documentation of agile software development does not encourage the documentation of the learning that takes place within agile software development teams. However, the utilisation of this knowledge on an organisational level would require some means of capturing the very valuable yet context-specific process knowledge of individual teams.

Even though the agile software development values, principles and practices have been the starting point for this research, the above differences, benefits, and challenges in both traditional and agile approaches to software development and SPI have been a guiding influence in this research.

4. Research Design

In this section, the research approaches and methods, and the evolution of the research are defined. This section contains a description of the research setting including the contexts of the research and a description of the case projects. In addition, the collection, storage, analysis, and reporting of the results of the research are described. Finally, the research design is summarised.

4.1 Research Methods and Evolution

This study can be classified as applied research and, more specifically, as constructive research. Järvinen (2001) defines constructive research as typically involving the building of a new innovation based on existing (research) knowledge and new technical or organisational advancements. Furthermore, Järvinen suggests that constructive research also involves an evaluation of the innovation. According to Järvinen, in constructive research it is possible to accept a prototype or even a plan as a research outcome instead of a final product. Following the classification of research epistemologies into positivist, interpretive or critical (Chua 1986), this study can be characterised as qualitative research that adopts an interpretive stance of investigation. The nature of interpretive research has been defined as based on the assumption that social constructions are the basis of our knowledge of reality and, rather than predefining dependent and independent variables, the focus is on the "complexity of human sense making as the situation emerges" while attempting "to understand phenomena through the meanings that people assign to them" (Klein & Myers 1999). In addition, this study can be claimed to adopt pragmatism as its underlying philosophy, as suggested is the case in most forms of AR (Baskerville & Myers 2004). In this study, the practical implications of SPI activities are considered vital components in understanding the meaning and truth and their effect on the theoretical implications.

In the research framework of March and Smith (1995), two types of science are identified: natural science and design science. From the viewpoint of this classification, this research falls into the category of design science as it "attempts to create things that serve human purposes" (March & Smith 1995,

p. 253) rather than trying to understand reality as such. As constructive research, the framework of March and Smith defines 'build' and 'evaluate' as the basic activities of design science, and distinguishes them from the 'theorization' and 'justification' activities of natural science. Järvinen (2001, p. 89) defines the building activity as "a process of constructing an artefact/innovation for a specific purpose", which consists of three steps: initial state, building process, and target state. In addition, Järvinen defines the evaluation activity as "a process of determining how well the artefact performs" (Järvinen 2001, p. 89) which requires metrics to support the assessment of the accomplishment. Within this research, both the build and evaluation activities were implemented in the case studies (Table 11).

Table 11. Research approaches and methods applied.

Research Approach	**Research Methods**	**Empiria of Evaluation (Case Project(s))**	**Evaluation of the Output**
Evaluate	Case study research	I	Paper II
Build-Evaluate	Action research – Case study research	II (and I)	Paper III
Build-Evaluate	Action research – Case study research	IV (and I–III)	Paper IV
Build-Evaluate	Action research – Case study research	V (and I–IV)	Paper V
Build-Evaluate	Action research – Case study research	V (and I–IV)	Paper VI
Build-Evaluate	Action research – Case study research	VI (and I–V)	Paper VII

In the first case project, the evaluation was conducted based on research data gathered from a finished project. From the second case project, the building-evaluation activity was used iteratively, complying with the cycles of agile software development of the case projects. The evaluation of the results occurred in more irregular cycles and was published in scientific papers (Table 11). It should be also noted that, when available, the evaluation activity always included the empiria from multiple case projects (Table 11).

The research framework of March and Smith (1995) identifies four types of design science products: concepts (i.e., constructs), methods, models, and

implementations (i.e., instantiations). In this research, two types of research outputs can be identified: methods and models. March and Smith define the *method as* "ways of performing goal-directed activities" (1995, p. 253) or, alternatively, as "a procedure or process for attaining an object" (Merriam-Webster Online Dictionary 2006). The project level SPI method (Iterative Improvement Process) (Paper V), which was iteratively designed during the research and the framework for deploying agile practices (Paper VI), can be considered as such outputs. In addition, March and Smith define *model* as a higher order construction used for describing tasks, situation, or artefacts, whereas Järvinen (2001) defines model as "a set of propositions or statements expressing relationships among constructs". Within this research, the integration of a project level SPI method for agile software development in the context of organisational SPI can be regarded as such a construction (Paper VII).

Figure 3 illustrates the evolution of the research. Seven main stages of research can be identified: a literature review, a case study (case project I), and five main AR cycles (case projects II–VI). In addition, each of the AR cycles consists of multiple internal iterations described in more detail in 4.1.4.

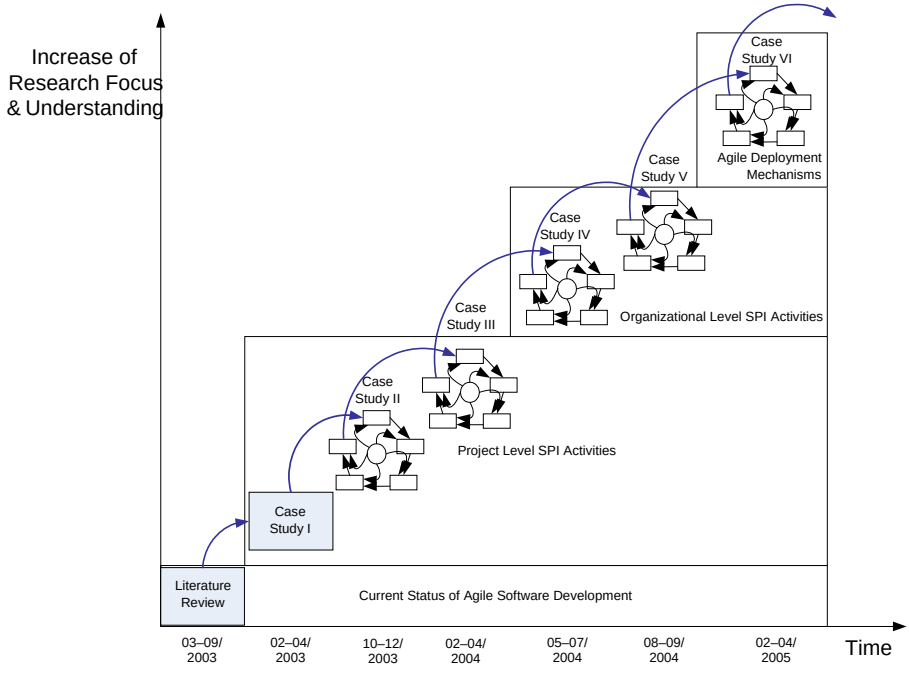

Figure 3. Evolution of Research.

.

A literary review of agile methodologies conducted in 2002 can be regarded as the starting point of this research. The review provided an important input for the research by offering knowledge of the current state of agile methods at that point in time. The six case projects of this research were conducted between spring 2003 and spring 2005 (Figure 3). These six case projects form a multiple-case study. According to Yin (2003), multiple-case studies enable broad generalisations based on the collective case study evidence. The consecutive arrangement of the case projects also enabled a longitudinal case study, which allowed the evolvement of certain phenomena to be studied over time (Yin 2003). Furthermore, the case studies of this research can be regarded as explanatory studies as they focus on "operational links needing to be traced over time, rather than mere frequencies or incidence" (Yin 2003, p. 6).

The focus of the study was adjusted throughout the research process based on the increased understanding of the topic. The AR approach in five of the case projects (II–VI) (sub-section 4.1.4), in particular, provided an opportunity for the researcher to iteratively re-adjust the focus of the research based on the results of the previous cycle. Figure 3 illustrates the study of existing literature, which was emphasised at the beginning of the research but which was also carried out as a continuous activity all through the research project. The three first case projects focused mainly on the project level SPI activities of agile project teams. Once the importance of organisational level SPI activities in the agile software development context was realised, the focus of the study was extended to this area (case studies III–VI). Finally, also the mechanisms of deploying agile software development were addressed in case project VI. It should be noted that the research of organisational SPI mechanisms provided new knowledge also from the viewpoint of project level SPI among the agile project teams. Thus, the focus of research was continuously expanded rather than changed.

In the following sub-sections, the different stages of this research are discussed in more detail. Firstly, the literature review is addressed; secondly, the first case study (case project I) is presented and, finally, the AR approach, as applied in case projects II to VI, is discussed.

4.1.1 Literature Review

The first preparatory stage of this research was conducted in 2002 as a literature review on agile software development. As suggested by Cooper (1984), the literature review aimed at defining the research topic's current state of knowledge. More specifically, the research aimed to identify the fundamentals of agile software development and to study the existing agile software development processes as well as to chart the available empirical evidence on the topic – scientific as well as anecdotal. The results of the literature review were published in (Abrahamsson et al. 2002).

According to Cooper, the cumulative nature of science requires trustworthy accounts of past research to form a necessary condition for orderly knowledge building (Cooper 1984). In this research, the literature review contributed to determining the focus of the incipient research in the field of agile software development, and developing insightful research questions on the topic, as suggested in (Yin 2003). The literature study revealed, among other issues, that the evidence on agile methods was, at the time, based largely on anecdotal rather than scientific evidence. Concurrently, this observation was also made elsewhere (Lindvall et al. 2002). Furthermore, an important input for the topic of this thesis was the finding from the initial literature review that only a few methods suggested any means for conducting SPI within agile project teams, even though such an activity was highly rated in the principles of agile software development (Agile Alliance 2001) and had been recognised by agile proponents. One of the twelve principles of the agile manifesto (Agile Alliance 2001) states that "at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly". Also Cockburn – one of the original signatories of the agile manifesto – has stated that "each situation calls for a different methodology" (Cockburn 2002). Furthermore, the XP practice of "just rules" declared that "the team can change the rules at any time as long as they agree on how they will assess the effects of the change" (Beck 1999). However, the existing agile methodologies, including XP, did not seem to provide a means for accomplishing such an iterative improvement. The preliminary study on agile methodologies revealed two existing methods that were to be an important starting point of the SPI research described in this thesis: the methodology-growing technique (Cockburn 2002) by Cockburn and a learning mechanism called postmortem reviews by Dingsøyr and Hanssen (2002). The empirical

evidence on SPI in the context of agile software development was still nearly non-existent (Dingsøyr & Hanssen 2002). Thus, it was clear that research was needed in the field of SPI in agile software development.

4.1.2 Case Study

The research method applied in the case projects of the research can be categorised as a case study. As suggested by Yin (2003), the research focused on a contemporary set of events and it relied on multiple sources of empirical evidence, both qualitative as well as quantitative.

The research method of the first case project also included elements of the history method (Yin 2003) as the researcher did not have any virtual access or control over the events, but rather the main source of research was the documentation of a finished project (Yin 2003). For example, a transcribed interview and data from the project retrospectives (Dingsøyr & Hanssen 2002) were available to the researcher; these had been collected by other stakeholders prior to the involvement of the researcher. However, the researcher did not completely rely on the past as there were relevant persons available to report on the events, as was recommended for a distinctive contribution of a history (Yin 2003). Even though the researcher had no opportunity to observe the events, she still had the opportunity for retrospective discussions with the software developers as well as other stakeholders of the project. The developers even participated in the publishing of the results in paper II.

4.1.3 Action Research

From the second case project onwards, the AR method was adopted. AR has been identified as constructive (Järvinen 2001) and qualitative research (Avison et al. 1999), which is one form of case study (Cunningham 1997). Unlike other types of case studies, the purpose of the AR case study is to develop "concepts which help facilitate the process of change" and where "theory emerges in the process of changing" (Cunningham 1997, p. 403). Furthermore, the AR process incorporates both building and evaluation sub-processes (Järvinen 2001) that are identified in the research framework of March and Smith (1995).

AR has been argued to provide highly relevant research results due to its foundation on practical action, which aims at solving an immediate problem situation, while also contributing to the theory (Baskerville 1999). AR originates from the field of social sciences in the 1940s (Lewin 1946). Since then AR has been suggested to solve the practical problems in the fields of organisational science (Susman & Evered 1978) and education in which it has also been accepted as a valid research method (Baskerville & Myers 2004). The importance of AR has also increased in IS (Information Systems) studies towards the end of the 1990s (Lau 1999) even though it is still largely ignored (Avison et al. 1999). Actually, it has been arqued that, still today, the "IS researchers continue to struggle to make excellent research practically relevant" (Baskerville & Myers 2004, p. 329) while acknowledging the potential of AR methods in providing a means to improve the practical relevance of their research (Baskerville & Myers 2004).

In the 1970s Susman and Evered identified a crisis in organisational science; while the practical problems of members, groups, organisations or networks of organisations were partially solved by using research methods that where able to generate knowledge to improve the effectiveness and efficiency of organisations, this was often accomplished at the expense of the quality of the working life of their employees (Davis & Taylor 1972). This was caused by highlighting the neutrality of how knowledge was created and, as a result, the underlying meanings and latent values were often not recognised (Susman & Evered 1978). Thus, the AR method aims at providing a means to perceive the interaction between the human social systems and information technologies as a whole entity, the parts of which affect each other (Baskerville 1999). In addition, AR aims at contributing to both science and practice (Rapoport 1970, p. 499) through iterative change and reflection (Susman & Evered 1978). On the one hand, it aims at contributing to the science "by joint collaboration within a mutually acceptable ethical framework" (Rapoport 1970, p. 499). On the other hand, AR is concerned with contributing "to the practical concerns of people in an immediate problematic situation" (Rapoport 1970) in a given social context. According to Susman and Evered (1978), one aim of AR is also to develop competencies of self-help for people facing problems.

AR has been defined to always involve a team that includes researchers and subjects as co-participants in the enquiry and change experiences (Baskerville

1999). Through interaction and personal understanding, the researcher also becomes part of the study (Baskerville 1999) while also adopts a helping role with practicioners (Baskerville & Myers 2004). The application domain of AR ideally includes active involvement of the researcher with expected benefit for both research and practice, immediately adopting the obtained knowledge, and a cyclical process where theory and practice are linked (Baskerville 1999). Such an approach seems to be extremely well suited for the domain of this research, in which the iterations of agile software development were used to evaluate and build SPI mechanisms. At project level, the collaboration of the researcher and the software developers aimed at providing mechanisms for immediate improval of the daily working practices of developers, while gaining research knowledge on how to conduct agile process adaptation among agile software development teams.

At an organisational level, the co-operation of the researcher and the Software Engineering Process Group (SEPG) aimed at improving the organisational software development process while gaining research knowledge on how to integrate agile software development and organisational SPI mechanisms. The project level improvement workshops, i.e., Post-Iteration Workshops (PIW's) at project level, the SEPG meetings at organisational level and the role of the researcher as a facilitator in both provided the researcher with an opportunity to iteratively intervene, influence, support and take part in the defining and planning of the actual process improvement actions and building of SPI mechanisms along the way. Thus, the SPI processes within the case projects were conducted in co-operation between the researchers and the software developers, all of them being "mutually dependent on each other's skills, experiences, and competency" (Lau 1999, p. 154).

AR is not a single research method, but refers to a class of research approaches (Baskerville 1999). Lau (1999) has identified four streams of AR: action research (AR), participatory action research (PAR), action science (AS), and action learning (AL). Unlike in traditional AR, the practitioners in PAR are involved as both subjects and co-researchers and "solve problems themselves by setting their own research agenda, collecting and analyzing the data, and controlling overuse of the findings" (Lau 1999, p. 150). In this research, however, the practitioners' focus was on solving the practical problems of their daily software development work. The purpose of the metrics data collection and analysis conducted by the software developers was thus also to provide SPI

knowledge for practical use. The researcher was able to utilise the knowledge for research purposes too. The emphasis of AS is on studying the "participants' behaviours as theories-in-use versus their beliefs as espoused theories" (Lau 1999, p. 150). In this research, however, the focus was not so much on the social behaviour of participants, but on how they put their knowledge and learning into action. AL varies from the traditional AR in the sense that its participants typically come from different situations and have been involved in different activities and face different problems. Thus, the learning goal in AL is individual rather than collective. In this research, however, the software developers worked in a single workspace and aimed at collective learning to improve their common working practices. Thus, in the AR categorisation of Lau, this research falls into the category of traditional AR.

Various frameworks have been suggested for evaluating the quality of AR studies. For example, Davison has defined a set of principles for canonical AR (Davison et al. 2004), and Hult and Lennung propose a set of major AR characteristics (Hult & Lennung 1980). A framework for evaluating AR studies especially in the context of IS studies has been proposed by Lau (1999). In the following sub-sections, the characteristics of this research are defined using Lau's AR framework and its four AR dimensions: 1) conceptual foundation, 2) study design, 3) research process, and 4) role expectations. The goal is to provide a definition of how AR was actually applied within the last five case projects of this study. It should also be noted that, in the following sub-sections, the criteria of each dimension is defined with reference to the respective parts of this thesis where the issue is defined in more detail.

4.1.3.1 Conceptual Foundation of AR

The dimension of "conceptual foundation" is first defined to include the research aim as well as the theoretical assumptions in order to provide the intellectual framework for the research. Furthermore, the criteria of perspective/tradition aims at defining the researcher's philosophical stance while the stream of AR is used to distinguish the intent of the study (Lau 1999). In Table 12, the "conceptual foundations" of this research are defined using Lau's framework.

Table 12. Conceptual Foundations of AR in the Research.

Criteria	Classification/Evaluation	Criteria in this Research	Further Definition
Research aim or question	Is the research aim or question authentic and practical in addressing a practical problem in an immediate situation?	– A new SPI method for conducting process adaptation (SPI) among agile software development teams – A tentative model for integrating traditional SPI of organisational level and agile process adaptation of software development teams – Integrating the agile process adaptation mechanisms as a part of an agile specific deployment framework (to support the deployment of agile software practices in organisations)	Papers II, III, V Papers IV, VI Paper VII
Assumptions	Is some form of theory, theme or concept included?	The underpinning of the research is in the theories and concepts of SPI and traditions of software engineering.	Underlying theories and concepts are defined in section 2.
Perspective /tradition	What is the adopted investigative stance?	Interpretive investigative stance Pragmatism as underlying philosophy Explanatory case study	Section 4.1 Section 4.1 Section 4.1.2
Stream	What AR type is used (AR, PAR, AS or AL streams) and is it described consistently?	This research is defined to follow the AR stream.	Section 4.1.3

From a theoretical perspective, the aim of this research is to build SPI methods for agile software development teams and organisations applying and deploying agile methodologies. From a practical viewpoint, the goal is to provide software development teams and organisations with mechanisms for identifying and resolving immediate problems faced both by software developers and software development organisations, and for improving their (daily) software development practices. Thus, the underlying concepts of this research can be found in the theories of SPI and in both the traditional and agile software engineering methodologies (defined in Section 2 of this thesis). The research design of this study is presented in Section 4 along with the perspectives, traditions, streams and methods adopted in the course of the research.

4.1.3.2 Study Design of AR

The dimension of "study design" describes the methodological details of the study (Lau 1999). In Table 13, the multiple criteria proposed by Lau for defining AR "study design" are presented along with a description of their use within this research.

Table 13. Study Design Dimension of AR within this Research.

Criteria	Classification/Evaluation	Criteria in this Research	Further Definition
Background	Does the background information provide a sufficient understanding of the research context? – organisation of research – nature and extent of its problems/needs	The research contexts of this research are defined	Section 4.2.1 Papers I, VII
		The six case projects of the research are defined	Section 4.2.2
		Various kinds of organisational needs defined: – Improvement of daily working practices of software developers – Improvement of organisational software process – Deployment of agile software development methods	Section 4.1.4.2
Intended change	What is the nature and extent of planned change?	Details of the change were not defined in advance, but the planning and implementation of change were done iteratively in co-operation among the respective organisational participants: researchers, software developers, management, and SEPG.	Section 4.1.4.2
Site	Is the involvement of site(s) and organisation defined?	Six individual and consecutive case projects were conducted in various organisational settings: laboratory, close-to-industrial, and industrial.	Sections 4.2.1, 4.2.2

Criteria	Classification/Evaluation	Criteria in this Research	Further Definition
Participants	Who are the participants?	The relevant stakeholders of the research and their roles within the SPI activities were defined.	Section 4.2.2
	Are the participants directly affected by the problem and benefiting from the intended changes to be made to address that problem or need?	– The project level SPI mechanisms developed to address the problems identified by the software development teams supported these teams directly in adapting/improving their daily working practices iteratively. – The developed organisational SPI mechanisms directly supported the organisational improvement of the organisational agile software development process/the organisational deployment of agile practices	Papers II–VII
Data sources	Is the data credible, dependable and confirmable?	Multiple data sources and data collection mechanisms were used and defined.	Section 4.3
		The validity of the research was evaluated using Yin's framework.	Section 4.1
Duration	Is there enough time for problem diagnosis, action intervention, and reflective learning to take place?	The timeline of the research, the schedule of the case projects and also the number and length of the project iterations were defined. The research was pre-defined on a case-to-case basis.	Section 4.1: Figure 3 Section 4.2.2: Table 19
Degree of openness	Is the process conducted as planned or will it evolve over time?	The AR process was defined on a case-to-case basis in an evolutionary manner.	Section 4.1
Access/exit	Are the intended type, level, and extent of access to the organisation defined?	The entry at the launching of each project/duration and the schedule of case projects were defined. The access of the researcher was arranged through the role of the researcher as a facilitator of SPI activities. The occurrence of SPI activities was defined for each case project.	Section 4.2.2: Table 19
	Is the exit point of the study defined?	The role of the researcher as a facilitator was pre-defined along with the exit at the end of a pre-defined project life-cycle.	Section 4.2.2
Presentation	Does the reporting provide sufficient information for judging its quality? What is the reporting style?	The central results of the research have been published in various scientific forums and have also gone through an appropriate review process.	Section 4.5

As illustrated in Table 13, the design of the research is defined throughout section 4 of the thesis. As AR is regarded as one form of case study (Cunningham 1997) the research design follows Yin's categorisation of case study stages (Yin 2003).

4.1.3.3 Research process of AR

The "research process" dimension is identified as the sequence of steps through which AR is conducted (Lau 1999). The sequence of steps for conducting AR makes it distinct from other research methods (Lau 1999). AR has been defined to consist of one or more iterations of problem diagnosis, action interventions and reflective learning (Lau 1999). Susman and Evered (1978) have also proposed an iterative cycle of AR, including the steps of diagnosis, action planning, action taking, evaluation, and specifying learning. In Table 14, the criteria of the "research process" dimension are defined.

Table 14. Research Process Dimension within the Research.

Criteria	Classification/ Evaluation	Criteria in this Research	Further Definition
Problem diagnosis	Are practical problems or needs identified?	Practical problems of software development iteratively identified and used as a basis for SPI activities.	Section 4.1.4.1 (Diagnosing)
Action interventions	Are planned and implemented actions identified?	Planned and implemented SPI actions iteratively identified (as a part of the adopted and developed SPI method)	Section 4.1.4.2 (Action planning) Section 4.1.4.3 (Action taking) Paper V
Reflective learning	Are reflections identified and explicit?	– SPI actions iteratively validated among the case project teams – SPI methods iteratively updated – Results published throughout the research in scientific forums	Section 4.1.4.4 (Evaluating), Section 4.1.4.5 (Specifying learning), Table 18 Papers II–VII (Section 4.5)
Iteration	Is there an iterative process planned as part of the study?	Five case projects of AR consisting of multiple iterations: Case project II: 5 iterations of AR Case project III: 5 iterations of AR Case project IV: 5 iterations of AR Case project V: 5 iterations of AR Case project VI: 4 iterations of AR	Section 4.1.1 Section 4.2.2
General Lessons	Are there general lessons from the study?	General lessons derived and published in scientific forums.	Papers II–VII (Section 4.5)

This research consists of five AR case projects, each of which can be regarded as one cycle of AR. Furthermore, each case project embodies several iterations of agile software development processes, each of which can also be considered a cycle of AR. Thus, the steps of problem diagnosis, action intervention, and reflective learning occurred iteratively during the individual case project as well among the individual case studies. The two-level cycles of AR in this research are defined in more detail in section 4.1.4 using the AR steps of Susman and Evered (1978). Furthermore, the general lessons of this research have been published in various scientific forums in order to submit the empirical evidence, its analysis and conclusions for review and feedback.

4.1.3.4 Role Expectations of AR

Lau suggests the dimension of "role expectations" to clarify how the researcher and other participants of the study are involved and what their capacities and expectations are (Lau 1999). In Table 15, Lau's criteria of role expectations in AR are presented in the context of this research.

Table 15. Role Expectations Dimension within the Research.

Criteria	Classification/ Evaluation	Criteria in this Research	Further Definition
Researcher	What is the role of the researcher?	Researcher in the role of a facilitator of SPI activities at project and organisational levels.	Section 4.2.2, Table 20
Participants	What is the role of participants?	Depending on the case project, the research included various participants, of which the software development teams were in a central role. The participants and their roles in the research are described in detail in section 4.2.2.	Section 4.2.2, Table 20
Competency	What improvement in competency is planned for participants?	– The competency of software development teams in iteratively self-improving their daily working practices. – The competency of organisational SPI stakeholders in supporting the process adaptation of agile software development teams and learning in a bottom-up manner. – The software developers identified the improvement goals of their software development iteratively	
Ethics	What ethical issues need to be addressed?	– An open manner of reporting the problems among the software development teams enhanced in PIW's. – No traceability provided between negative findings of software process and the origin (an individual software developer) beyond a PIW session/software development team. – The confidentiality issues agreed upon.	

As in an emergent form of AR (Lau 1999), the researcher was in a role of facilitating the SPI activities of the case project teams. The various participants of the different case projects are defined in more detail in section 4.2.2. In a central role, however, were the software development teams iteratively conducting the activities of problem diagnosis, action interventions, and reflective learning, which were built on the mechanisms of the Iterative Improvement Process of SPI (Paper V). Accordingly, they were iteratively dealing with and addressing the practical problems of software development, thus aiming at improving the organisational base process to better fit their context specific needs. On the other hand, the organisational stakeholders were adopting SPI mechanisms in order to support the project teams in their SPI activities while they were also systematically dealing with the SPI knowledge of

software development teams in order to improve the base process and practices. Thus in the context of this research, competency refers to the ability of the organisational stakeholders to improve the working practices and, as a result, to increase the efficiency of the production of software and the satisfaction of software developers in their daily working practices.

The ethical issues addressed within the research concerned, firstly, the non-transparency between individual software developers and the resulting SPI activities among software development teams. Secondly, the issue of confidentiality was discussed and agreed upon among the research and customer organisations of the research.

4.1.4 Five Cycles of Action Research

Susman and Evered (1978) have identified a cyclical and iterative process of AR that consists of five steps: 1) diagnosing, 2) action planning, 3) action taking, 4) evaluating, and 5) specifying learning.

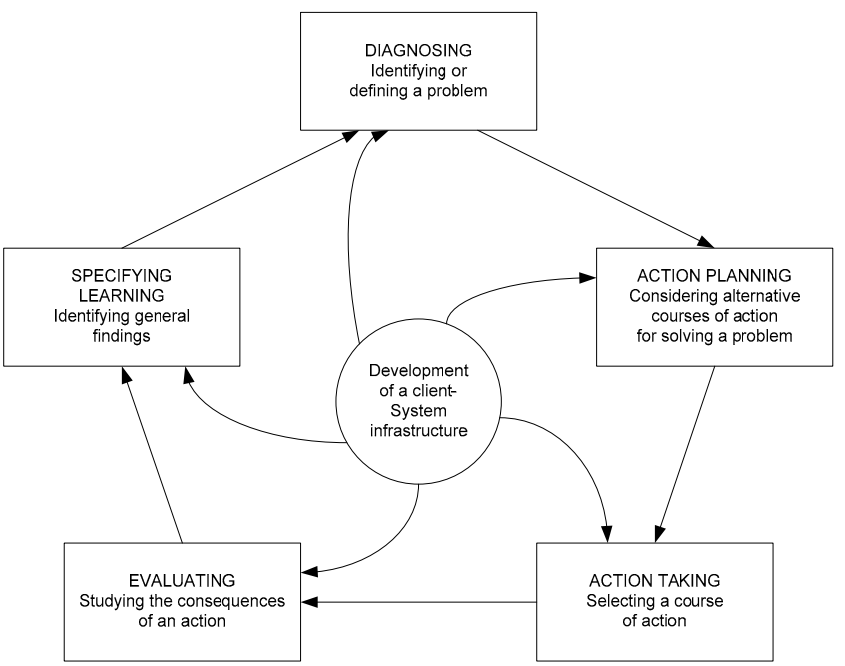

Figure 4. Susman and Evered's Cyclical Process of AR (Susman & Evered 1978).

The immediate client-system infrastructure (Figure 4) of this research can be regarded as that of a software product being developed, but also as a software development process that was collectively evolved by the software developers, SEPG of the research organisation and the researchers. However, whereas the software developers adapted (i.e., tailored) the underlying software development process to fit their context specific needs in PIW's, the SEPG meetings were held to enhance long-term improvement of the base process (Mobile-D™) (Abrahamsson et al. 2004, Ihme & Abrahamsson 2005).

In this research, two different levels of the AR cycle can be identified: 1) the agile software development iterations within each of the case projects, and 2) the six cycles of case projects conducted in the continuum of the research. The first type of AR cycle involves the software developers and SEPG team members as practitioners and the researcher as facilitator of the SPI meetings of the two stakeholder groups. In fact, the AR cycles within a case project are run concurrently with the cycles of the Iterative Improvement Process cycle (i.e., the cycle of conducting PIW sessions). In the latter, the developers aim to adapt and improve their daily working practices with the help of the facilitator (the researcher acting as a facilitator), and the researcher gains knowledge for the research purposes. The steps of the Iterative Improvement Process are defined in detail in Paper V. Part of the AR cycles also included in the iterative organisational SPI activities of the SEPG team that also involved the researcher as a facilitator.

The second type of AR cycle involves the researcher's activities to analyse data from the previous case project(s) in order to further improve the SPI mechanisms for the next project and, then, to evaluate the effects of the changes afterwards. Each case project provided an AR loop in which the focus of research could be readjusted based on the results of the previous iteration. The short iterations of agile software development during the case projects also provided an opportunity for the researcher to rapidly take and evaluate actions concerning, for example, changes in the evolved SPI method in collaboration with the software development teams and SEPG.

The central focus of this research – SPI both within project teams and at organisational level, and the evolvement of the SPI mechanisms for project and organisational purposes – involve the six central characteristics of AR identified

by Susman and Evered (1978): 1) orientation towards creating a more desirable future for people dealing with practical concerns, 2) collaborativeness between the researcher and the client system, where the research process will benefit both parties, 3) the focus on generating convenient procedures for communication and problem-solving and its modification for the relevant environment, 4) generation of theory grounded in action in cyclical process (Figure 4), where the theory benefits the action and vice versa, 5) the re-examination and reformulation of the theories and prescriptions in every new research situation, and 6) the realisation that the events and relationships between people, for example, are functions of each situation yet often invariant. Thus, it could be argued that the AR method and the focus of this research represent an ideal match, as has also been suggested more generally in IS research and AR (Avison et al. 1999).

In the following section, Susman and Evered's (1978) AR steps are defined in the context of this research.

4.1.4.1 Diagnosing

In the diagnosis of the problems and goals of improvement in this research, the practice informed the research and vice versa, a unique characteristic of AR as suggested by Avison et al. (1999). Two cycles can be identified in which the activity of "defining of the problem" was conducted: between case projects and iteratively during the project. First, by analysing the research data from the finished and previous projects, the researcher was able to reconsider the focus of the research while identifying weaknesses in the adopted SPI methods. Secondly, the software development teams (in PIW's) and the SEPG groups (in SEPG meetings) together with the researcher (i.e., facilitator) iteratively defined the problems and the possible solutions concerning both project specific and organisational software development process. The improvements also concerned the SPI mechanisms adopted within the project. In Table 16, the different diagnostic activities, their occurrence, immediate target and also the participatory actors and utilised inputs for the diagnosis are summarised.

Table 16. Diagnosing Activities in the Case Projects.

Diagnosis During the Case Projects				
Activity	Occurrence	Target of Improvement	Actor(s)	Input
PIW's (experience collection step)	4/4/4/4/3	**Practice:** Project specific software development process/effectiveness of the software development/ motivation of software developers **Research:** Project level SPI method in agile software development	Development Team, Researcher, i.e., facilitator	Experiences Knowledge Feedback Metrics
SEPG meetings	0/1/6/4/1	**Practice:** Organisational software development process/Organisational SPI mechanisms **Research:** Integrating SPI method of agile project teams and organisational SPI practices	SEPG team, Researcher, i.e. facilitator, (software developers)	PIW results Postmortem results Metrics Experience Knowledge
Diagnosis Between the Case Projects				
Activity	Occurrence	Target of Improvement	Actor(s)	Input
Data Analysis	5 (II–VI case projects)	**Research:** Research focus/Project level SPI method and organisational SPI model **Practice:** Base SPI method and activities for future project and SEPG team	Researcher	PIW data Postmortem results Metrics SEPG meeting data Experience Knowledge

During the five AR case projects, a total of 19 PIW's were iteratively held among the software development team with the researcher acting as a facilitator. The field "occurrence" in Table 16 defines the number of PIW's and other SPI activities within the different AR case projects. The PIW's were placed as the first activity of each software development iteration. In practice, the goal of the PIW's was to involve the software developers in identifying the problems in their daily working practices (experience collection step) in order to accomplish improvements and, consequently, to increase the effectiveness of the software development process and its appeal to the software developers. In the SPI

processes, the knowledge and experience of the developers were in a central role, while the metrics data was also utilised.

Between the case projects, the researcher analysed the data from the previous and preceding projects. The goal was to diagnose – on a case-by-case basis – the future direction of the research as well as to define the next versions of the research outputs. From the viewpoint of practice, the diagnosis between the case projects resulted in a revised SPI method to be adopted in the next project team, and also enhanced organisational SPI mechanisms for the SEPG group.

4.1.4.2 Action Planning

The selection of alternative actions to be taken was in a clear continuum with the activities of diagnosis (sub-section 4.1.4.1). Thus, the action planning was done between case projects by the researcher and iteratively during the project in collaboration with the software development teams (in PIW's) and SEPG group (in SEPG meetings) (Table 2). During the case projects, the planned actions concerned both project and organisational level SPI activities. The implications for practical project-level action planning focused on the adaptation of the software development process adopted by the development team, also including the SPI method itself as a part of the software development process. On the organisational level, the action planning within the SEPG team aimed to improve the underlying base process (i.e., Mobile-D™) based on the improvement opportunities identified by the project team (in PIW's and project postmortems), measurement data and the process (improvement) knowledge of SEPG team members.

Between the case projects, the action planning was concerned with the definition of the research focus for the following case project. In addition, based on the diagnosis, the action planning also focused on how the SPI method adopted in the previous case project should be improved for the next project team, and how the organisational SPI mechanisms should be readjusted for the following case project.

The overall action plan of the research had a dual goal: to provide added value for both research and practice. From the research viewpoint, the goal was to gain understanding and knowledge of SPI within the projects and organisations conducting agile software development in order to build and further refine the

existing SPI mechanisms in this specific context. In practice, the more immediate goals were: 1) to improve the effectiveness of software development in agile software development projects, 2) to improve the satisfaction of software developers in their daily work, 3) to provide mechanisms to improve the organisational agile software development process (Mobile-D™), and 4) to support the organisation in deploying the agile software development process.

4.1.4.3 Action Taking

Baskerville (1999) defines the action-taking step as involving the collaboration of researchers and practitioners in an active intervention, in which certain changes are made. In this research, the action-taking during the software development projects concerned the implementation of the process improvements identified, agreed and defined in collaboration with the software developers and the researcher. The action-taking occurred in the iterative cycles of agile software development. As the improvement actions concerned the working practices, the action-taking, logically, took place during the software development activities. The same iterative cycles of action-taking were conducted at the organisational level SEPG meetings (case projects III–VI), where the organisational level SPI actions were identified. In Table 17, the number of actions taken by the software development and SEPG teams of each case project is presented.

Table 17. Actions Taken in the Case Projects.

Level of Action	Case Project	Number of Identified Actions	Main Actor
Project Level	I	16	Software Development Team
	II	56	
	III	33	
	IV	27	
	V	26	
	VI	24	
	Total	**182**	
Organisational Level	III	25	SEPG
	IV	57	
	V	30	
	Total	**112**	

In total, it was agreed that 182 actions would be conducted at project level within the case project teams. This number includes the actions related to the improvement of *practice* within the case projects and, thus, it does not include the actions that were taken by the researcher on the SPI methods in between the case projects. At project level, two types of actions could be identified: 1) actions directly related to SPI issues such as working procedures and tools of software developers, and 2) actions related to other concerns of developers such as acquisitions concerning the working environment. The SPI actions jointly identified by the developers and the facilitator (i.e., researcher) also included improvements directly-related to the SPI method itself as it was part of the software development process. The project level actions were carried out by the software development team, yet a proportion of the actions also required the support or action-taking from other organisational stakeholders, such as the support team.

At the organisational level, a total of 112 actions were taken by the SEPG. The number includes data from case projects III–V during which the organisational SPI methods were actively built and conducted. Furthermore, the data of the organisational SPI activities of case project VI is excluded due to the different role of the researcher in that context. The organisational action-taking considered several issues that were identified largely by analysing the SPI actions of the project teams. The organisational level improvements concerned, for example, training, software development tools, data collection mechanisms, the underlying software development process model (Mobile-D™), and support of project teams in their SPI activities during the projects.

In AR, the role of an action researcher is defined as being active in discovering improvements, as well as in controlling that the improvements are properly applied (Chein et al. 1948). The actions taken during this research included, for example, the building and evaluation of follow-up and validation mechanisms for the project and organisational SPI methods in order to provide a means, for both practice and research, to evaluate whether the actions were actually taken and how successful they were (see Papers IV, V, and VI).

4.1.4.4 Evaluating

In AR, once the actions are completed, the researchers and practitioners should collaboratively evaluate the outcomes (Baskerville 1999). In the case studies, the iterative cycles of agile software development provided an adequate setting for evaluating the process improvement actions of software development teams. In fact, the Iterative Improvement Process (Paper V) has been built to include an activity of follow-up and evaluation (see Paper V), in which the software developers and the facilitator systematically assess how effective the improvements are and if they should be embedded in daily working procedures or disregarded. These activities of follow-up and evaluation can be considered as evaluating the implications of AR *in practice*, i.e., assessing whether the software developers are provided with adequate support and mechanisms for actually improving their software development process and whether the effects are considered as positive.

On the other hand, the implications of AR *for research* had to be evaluated. In this research, the active participation in the SPI activities provided the researcher with immediate feedback from software developers and experience on whether and how the SPI mechanisms needed to be modified. Thus, the researcher was continuously and iteratively evaluating the SPI methods as well as the effects of the changes made to them during the case projects (after each PIW and SEPG meeting) and also between case projects based on the research data gained from the project.

4.1.4.5 Specifying Learning

Baskerville (1999) considers "specifying learning" as an ongoing process, in which three targets for disseminating the results knowledge gained in the AR can be identified: 1) organisation of research, 2) preparation for further AR intervention, and 3) scientific community. In the research, the specifying learning activities were, indeed, conducted in an ongoing fashion and with multiple targets. Firstly, from the viewpoint of the scientific community, the research findings were published in various scientific forums (see more in section 4.5) so as to report the results of the research in its different stages and to gain feedback from the scientific community. From the viewpoint of further AR intervention, the learning was also suggested by the researcher for use in

readjusting the research focus especially between case projects. In this task, the researcher needed to consider various alternatives for broadening and re-adjusting the research focus based on the experiences and research data gained form the previous project(s). As a result, a number of modifications to the research focus can be identified (Figure 3). The evolution of the SPI mechanisms, methods and techniques used during the case projects provides an overview of the learning that occurred during the case projects (Table 18).

Table 18. Evolution of the SPI Mechanisms in the Case Projects.

Case Project	Project Level: Iterative Improvement Process	Organisational Level: SEPG activities
I	• KJ method • Flap-sheet of action points	-
II	• Action point document • Systematic follow-up procedures • More exact action points	-
III	• Action point template • Quality team feedback in PIW's • Action points with responsibilities, schedule, and request for external support • Qualitative validation of process improvements	• Establishment of be-weekly SEPG meetings for systematic evaluation of org. level action taking • Support mechanisms for project team SPI activities
IV	• Be-weekly stand-up PIW • Quantitative validation of process improvements	• Quantitative feedback for project teams
V	• No alterations in the method	• Organisational metric set for analysing project level SPI
VI	• No be-weekly stand-up PIW's	• SEPG meeting at the end of the project

Table 18 illustrates the evolution of the 'specifying learning' activities throughout the overall project. Analysis of the research data between the case projects, and gathering direct feedback from the developers iteratively during the project, revealed opportunities for building SPI mechanisms and activities at both project and organisational levels. In Table 18, it can also be seen how the need to alter especially the project level SPI mechanisms diminished towards the end of the series of case projects.

Furthermore, from the viewpoint of the research organisation, the SEPG team analysed the learning of project teams in SEPG meetings in order to utilise it in addressing the relevant organisational issues, e.g., content of training, the organisational software development model and its description, software development tools, and data collection mechanisms. From the viewpoint of customer organisation, the learning was embedded in the end product being developed, in its quality, content, and the efficiency of its development process. Throughout the case projects, the customer was participating in the different software development activities where the product was defined (e.g., iterative planning) and evaluated (e.g., iterative customer testing). In case project VI, the specification of learning also involved the evaluation of the pilot project in order to specify how the Mobile-D™ process model would fit in the overall organisational context.

4.2 Research Setting

This research involves six case projects. A description of the research environments (4.2.1) as well as the case projects and their organisation (4.2.2) can be found in the following sub-sections.

4.2.1 Context of Research

The six case projects of this research were conducted in three different research contexts, which could be characterised as: laboratory setting, semi-industrial research setting and industrial research setting. The context in which both the laboratory and semi-industrial research was conducted, i.e., the ENERGI (Industry-Driven Experimental Software Engineering Initiative) research context, has been defined in detail in Paper I. Similar research environments for studies on agile software development have been, later on, proposed also elsewhere (Back et al. 2005).

4.2.1.1 Laboratory Research Setting

A preliminary study on agile methods (Abrahamsson et al. 2002) conducted at VTT revealed that, at the time of launching the research of this thesis, the

empirical evidence on the actual benefits and suitability of agile methods in different contexts was based largely on anecdotal evidence rather than on scientific knowledge. The urgent need to empirically assess the applicability of agile methods in a structured manner (Lindvall et al. 2002) was evident in order to benefit both the research and industry. Thus, the ENERGI environment was established at 2003 to further study, evolve, experiment, and empirically evaluate the proposed agile methods prior to their launching in an industrial context and employed in implementing real software products. In other words, ENERGI was designed to provide an environment for conducting software projects with both research and business objectives and to enhance the close collaboration of research and industry. In ENERGI, the research organisation provides the software development teams with the physical environment with office-space, equipment, training, support and, most importantly, the software development processes and methods. In return, the research organisation is provided with the benefits of extensive and ongoing research throughout the development process. On the other hand, the customer organisation is provided with the experiences of novel software development methods in addition to the actual software development product.

The first two case projects of ENERGI, however, were conducted in an environment that could be characterised as a laboratory setting (Figure 5). In other words, the end product was implemented for internal and research use without any external customer organisation being involved. In consequence, the internal customer of the research organisation was taking part in the development activities. In addition, the software development team was formed from university students in the final stages of their studies and inexperienced in using agile software development methods. The team was working in an office-space located in the facilities of a research organisation. The reason for conducting the first case projects in such a controlled setting was to gain a high degree of research control, which would allow the influence of certain variables, such as business pressure and influence of external stakeholders, to be eliminated. Thus, the aim was not so much to control the behavioural events or to "sample over the variables that are being manipulated" (Wohlin et al. 2000, p. 12) as in experimental research but rather to "sample from the variables representing the typical situation" (Wohlin et al. 2000, p. 12). The AR team, on the other hand, had an active role in training, supporting, and coaching the software development teams throughout the case projects while conducting

research. The researchers in the AR team were involved in the activities of quality and support teams (Table 20) and they all had their own special focus and field of know-how in the agile software development methods applied in ENERGI projects. As a result, multiple master's theses (e.g., Hanhineva 2004, Hulkko 2004, Kyllönen 2005) along with other scientific publications (e.g., Abrahamsson 2003, Hulkko & Abrahamsson 2005, Korkala & Abrahamsson 2004, Koskela & Abrahamsson 2004, Kähkönen & Abrahamsson 2004) have been published in the different areas of agile software development.

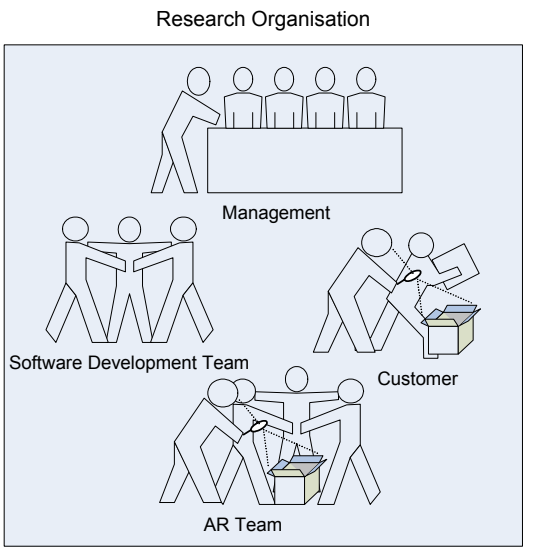

Figure 5. Laboratory Setting of Research.

4.2.1.2 Semi-Industrial Research Setting

In case projects III to V, the research was conducted in the semi-industrial context of ENERGI (Figure 6).

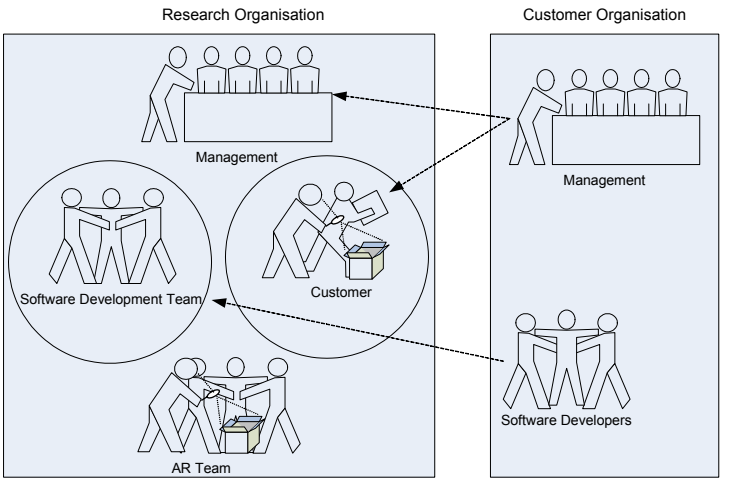

Figure 6. Semi-Industrial Research Setting.

While the software development was conducted in the premises of the research organisation, the participation of, and collaboration with, the customer organisation was extensive. Firstly, the customer organisation provided software development teams with their experienced software developers who also possessed valuable knowledge of the end product that was being developed for the customer organisation. Furthermore, the customer organisation provided the off-site customer for the case projects, who was actively participating in the defined activities of the development team. The development was managed in collaboration with the customer and research organisation. For more details, refer to Table 20 and sub-section 4.2.2.

4.2.1.3 Industrial Context

In case project VI, the research and the SPI methods were transferred to an industrial context (Figure 7). The customer organisation was interested in exploring the opportunities of agile software development, firstly in a pilot project. Thus, the Mobile-D™ (Abrahamsson et al. 2004) process that was developed within the five ENERGI case projects was adopted in the case project. The reason for selecting Mobile-D™ was its suitability regarding the mobile product that was to be implemented and the availability of expertise regarding the Mobile-D™ process and its practices. The goal of the customer organisation was to gain experiences concerning the software development process of the

pilot project in developing an organisation specific agile process model alongside the traditional process model.

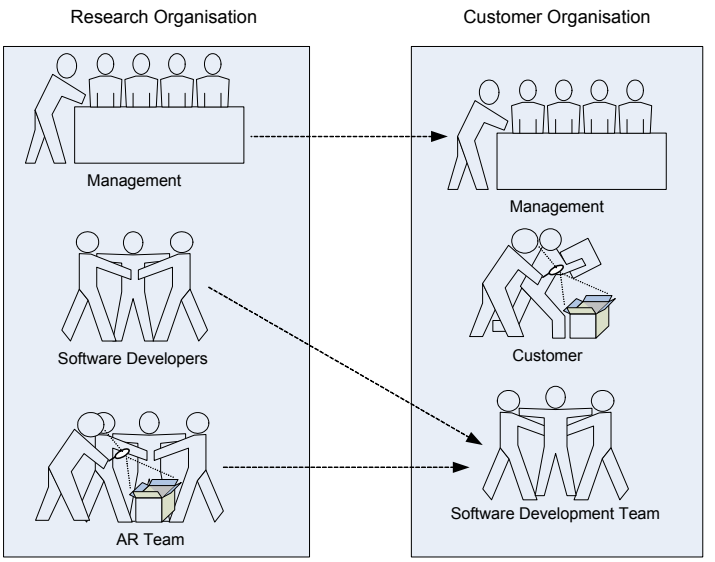

Figure 7. Industrial Research Setting.

The case project was conducted in the premises of the customer organisation. The software development team mainly consisted of the software developers of the customer organisation while it was also strengthened with members of the research organisation (i.e., AR team members and additional software developers) (see sub-section 4.2.2). Thus, the AR team of the research organisation was actively participating in the pilot project. Furthermore, the management of the pilot project was conducted in close collaboration between the research and customer organisations. The customer for the agile software development project was naturally provided by the customer organisation.

4.2.2 Case Projects and Organisation

Table 19 provides an overview of the characteristics of the six case projects.

Table 19. Characteristics of the Case Projects.

Characteristic	Project I	Project II	Project III	Project IV	Project V	Project VI
Context	Laboratory	Laboratory	Semi-Industrial	Semi-Industrial	Semi-Industrial	Industrial
Product	Intranet Application	Mobile Software	Mobile Software	Mobile Software	Mobile Software	Mobile Software
Iterations	6	6	6	5 (planned) 9 (actual)	5	5
Duration	9 weeks	9 weeks	9 weeks	9 weeks (p) 11 weeks (a)	8 weeks	8 weeks
Schedule	02–04/2003	10–12/2003	02–04/2004	05–07/2004	08–09/2004	02–04/2005
Development Effort in person months (pm)	7.5 pm	10 pm	5.5 pm	5.2 (total 9.1) pm	7.1 pm	7.2 pm
Project Level SPI activities	5 PIW's	4 PIW's 1 PMA	4 PIW's 1 PMA	4 PIW's 1 PMA	4 PIW's 1 PMA	3 PIW's 1 PMA
Organisational SPI activities	-	-	1 SEPG meeting	6 SEPG meetings	4 SEPG meetings	1 SEPG meeting
Process	XP	Mobile-D™ 0.1	Mobile-D™ 0.2	Mobile-D™ 0.3	Mobile-D™ 0.4	Mobile-D™ 0.4

The two first projects were conducted as laboratory experiments, case projects III–V as semi-industrial case projects, and the last project VI as an industrial case project (see sub-section 4.2.1.1). All the projects, except for the first one, focused on the development of mobile software for various mobile devices. Thus, all the case projects produced a real software product, even though the outputs of the first two projects were mainly built for research purposes. All the projects adopted an agile software development process model with several short iterations (Table 19). The software development process was incrementally built during and in between the case projects and evolved from XP to various versions of the Mobile-D™ process. The improvement of the underlying software development process model was conducted on two levels: at project level (i.e., iteratively within the development teams) and, in the four last projects, at organisational level. The methods of SPI were incrementally developed during this research and, altogether, 24 PIW's, 5 project postmortems, and 12 SEPG meetings were held in the course of this study (Table 19).

The focus of the PIW's was to provide the project team with mechanisms to adapt the base process of software development iteratively throughout the project. The main focus of the SEPG meetings was to improve the specific agile software development process of the given research context. In the SPI activities, metrics data and the process expertise of SEPG were utilised, while the process knowledge of the development teams (PIW's and PMA's) was also given a central role.

In the case projects, the SPI activities involved various stakeholder groups: software engineering process groups (SEPG) and support teams consisting of AR team members, software development teams, and customers, typically of agile software development (Beck & Andres 2004). The composition and the goals of the different stakeholder groups varied between the different projects (Table 20).

Table 20. SPI Organisation of Case Projects.

Case Project	SEPG	Support Team	Project Team	Customer
I	-	Management (1) AR team (4)	4	Research On-Site
II	-	Management (1) AR team (4)	~5.5	Research Off-Site
III	Support team Facilitator	AR team (7)	4	Industrial Off-Site
IV	Support team Facilitator	AR team (7+2 external)	4,5 (originally 6)	Industrial, Off-Site
V	Support team Facilitator	AR team (8 + 2 external)	4–6	Industrial, Off-Site
VI	Management (1) Facilitator (1) Architect (1) Development Team (2 internal + 3 external) Customer (1) QE team (3)	Management (1) External AR team (2) (as project manager and developer)	5 (+ 3 external QE team)	Industrial, In-house, Off-Site

The central tasks of the support team was to provide training for the software developers concerning the agile software development process that was adopted and to coach and support the project team throughout the project in the process, tools, methods, and the infrastructure of the project. Furthermore, the support team was responsible for providing the mechanisms for the metrics data collection and supporting the project level SPI activities among the software development team. Thus, one of the support team members had the role of a facilitator and participated and conducted the PIW's and PMA's among the development teams. The support team consisted of the researchers that formed an AR team in which everyone had specific expertise in agile software development and a specific focus of research. In addition, the last case project team was strengthened with a Project Manager and two software developers from the research organisation. The Project Manager and one of the software developers were also members of the agile AR team of the research organisation. Thanks to their previous experience and knowledge of agile methods, tools, and procedures, they were able to provide timely support for the rest of the project team, who had little experience of agile software development. A management representative also participated in some of the informal meetings of the support team, which were held during the first two case projects. Furthermore, in the last case project, the management provided continuous organisational support for the development team in deploying agile software development methods.

The project teams were in a central role regarding the SPI activities throughout the case projects. For one, they adopted the Iterative Improvement Process (Paper V) in which they iteratively conducted PIW's and PMA's together with the facilitator in order to systematically adapt their base process and to provide the organisational level with SPI requests and opportunities (Paper VI). In case project VI, the project team also included an external quality engineering team, which was responsible for the testing activities of software development (except for unit testing).

The SEPG was established at the end of case project III, once the organisational SPI activities had been established. In case projects III-V, the researcher took the role of a facilitator in the organisational SPI meetings. The rest of the SEPG team included support team members who had both experience and knowledge of certain aspects of the agile software methods used in the case projects. In addition, two external researchers participated in the SEPG team meetings in

case projects IV–V. In case project VI, the SEPG meeting was held at the end of the pilot project in order to improve the organisational agile software development process. In this SPI activity, the meeting was facilitated by the management, while the role of the external PIW facilitator was to provide SPI knowledge when needed. In addition, other stakeholders of the pilot project participated in the SEPG meetings (Table 20).

In all the case projects, the customer was in a central role by iteratively evaluating the quality of the end product. Occasionally, the customers also participated in the PIW's in order to provide their experience regarding, mainly, the product quality aspect rather than the software development process itself. Furthermore, in case project VII, the customer and the management took part in the PMA activity at the end of the project.

In the following section, the contents of the case projects are discussed in more detail.

4.2.2.1 Case Project I: eXpert

The first case project of this thesis was also the first project of the ENERGI initiative. Thus, at that time the research into agile methods in practice was in its early stages and also new to the researchers involved. For this reason, a decision was made to first start by building a software product for an internal customer and mainly for research purposes. In the first case project, the research focused on building an intranet application for managing research data. Consequently, the first project team was fully formed from experienced university students. It was also decided that the starting point would be to employ the XP (Beck 2000) method and its practices as a base process for the project. One reason for selecting XP was the fact that, at the time, XP was one of the most documented agile methodologies (Abrahamsson et al. 2002) and provided specific descriptions for the agile software development process.

The timeframe as well as the cost of the project were fixed prior to the project, yet the requirements of the product were flexible and subject to change throughout the project. The duration of the eXpert project was set at eight weeks and it took place between February and April 2003 (Table 19). In all, six software development iterations were conducted in the project. The first three iterations lasted two weeks

and the last three iterations one week each. The last iteration was concerned with system-testing and final fixing of the defects found in the product.

The software development team of eXpert consisted of four developers, all of whom were 5–6th year university students with 1–4 years of industrial experience in software development. None of the team members had any earlier experience in using agile methods. The project team was working fixed office hours (24 hours a week) in an open-office space throughout the project. In the project, an on-site customer was present in the open-office space at an average of 83% of the development time (Koskela & Abrahamsson 2004).

4.2.2.2 Case Project II: zOmbie

The second case project on the ENERGI case projects focused on the development of mobile software. This was yet another new territory for the researchers, for which reason it was essential to have a high degree of control and low degree of business pressure by implementing the end product for an internal customer. The goal of the software development team was to implement a service enabling mobile stock exchange.

Project II employed the first version of Mobile-D™ software development process. It was evolved on the practices of XP, which were enhanced during the project and thereafter to better suit the mobile software development context. From this project onwards, the central SPI goal of the ENERGI organisation was to develop an agile software development process model especially for mobile software development.

The timeframe as well as the cost of the project were fixed prior to the project, while the requirements of the product were flexible and bound to change throughout the project. The duration of the zOmbie project was set at nine weeks and it took place between October and December 2003 (Table 19). In all, six software development iterations were conducted in the project. The first iteration lasted one week, the second, third and fourth two weeks, and the last two iterations one week each. The last iteration was the system testing and fixing phase to detect and fix the defects in the product.

The project team consisted of five experienced university students (i.e., 5–6th year students in information processing science) and one software developer with several years of industrial experience, who also had a research interest in the project. He was also acting as a valuable project manager and an on-site coach for the project team due to his experience with both agile software development and the Mobile-D™ process, both of which were new to the rest of the development team. The project team was working 24-hour weeks in an open-office space.

4.2.2.3 Case Project III: bAmbie

In the third case project, the software product development focused on implementing a mobile software product as an extension to an existing software product of the customer organisation. Thus, the business pressure and the quality requirements for the end product were high. The software development process was conducted with the second version of Mobile-D™ and involved an active off-site customer from the customer organisation who was participating in the different activities of the development process, such as the planning of iterations.

The timeframe as well as the cost of the project were fixed prior to the project, while the requirements of the product where flexible and bound to change throughout the project. The duration of the bAmbie project was set at nine weeks and it took place between February and April 2004 (Table 19). The resulting end product consisted of 3800 (logical) lines of code implemented in a total of 5.5 person months. In all, six software development iterations were conducted in the project. The first iteration lasted one week, the second, third and fourth for two weeks each, and the last two iterations one week each. The last iteration was the system testing and fixing phase to detect and fix the defects in the product.

The project team consisted of an experienced software developer from the customer organisation as well as three experienced university students (i.e., 5–6th year students of information processing science). Furthermore, an off-site customer from the customer organisation was iteratively participating in certain activities of the software development, such as the planning game (Beck 1999). The project team worked 24-hour weeks in an open-office space.

4.2.2.4 Case Project IV: uniCorn

The focus of the uniCorn project was to produce a mobile software product for an external customer organisation as an extension of their existing software product. The business pressure, time constraints and quality requirements for the end product were, once again, high. The timeframe as well as the cost of the project were fixed prior to the project and the requirements of the product were expected to change throughout the project. Originally, the duration of the bAmbie project was set at nine weeks and it took place between May and July 2004 (Table 19). However, the uniCorn project faced personnel problems during the two first iterations, which resulted in changes in the team structure as well as in the duration and total effort of the project and the composition of the iterations. As a result, the planned project of nine weeks and five iterations (four two-week iterations and a one-week system test and fix iteration) evolved into a total of nine iterations and 11 weeks. After the first two iterations, the project was re-launched. The first two iterations consisted of 3.9 man months of effort and the seven one-week iterations consisted of 5.2 man months. In total, the effort of the software developers in uniCorn was 9.1 person months (see Table 19).

Originally, the project team consisted of an experienced software developer from the customer organisation as well as four experienced software developers, who were taking a one-year Symbian OS intensive course for professionals and doing their half-year training in ENERGI. The project team worked 24-hour weeks in an open-office space. The customer organisation, including the off-site customer and the software developer, was the same as in project III. Thus, one of the team members had earlier experience of Mobile-D™, agile software development and of the end product. At the end of the second iteration, the size of the project team diminished from six to four software developers. However, from the third iteration onwards, an experienced Project Manager (the Project Manager in case project II and software developer in case project I) joined the team, on a part-time basis (see Table 19).

4.2.2.5 Case Project V: Bubble

The focus of the Bubble project was to produce a mobile software product for an external customer organisation. The timeframe as well as the cost of the project

were fixed prior to the project. The duration of the Bubble project was eight weeks and five iterations (1 x 1 weeks 3 x 2 weeks, and 1 x 1 weeks) and it took place between August and September 2004 (Table 19). The project was conducted in a total of 7.1 person months. The size of the project team varied between four to six developers for the different iterations, including an experienced project team from the uniCorn project, which was complemented with an external TDD expert. The customer organisation also provided an off-site customer for the project.

4.2.2.6 Case Project VI: Phantom

In the Phantom project, the entire process model of Mobile-D™ – as evolved during the incipient case projects of ENERGI – was transferred for piloting in an industrial context. Accordingly, the Iterative Improvement Process (Paper V) was adopted. The researcher took part in the project in the role of facilitator in the project level SPI activities.

The project was conducted in the premises of the case organisation and focused on the development of a mobile security product. The timeframe as well as the cost of the project were fixed prior to the project. Some central requirements as well as the architectural structure of the product were fixed beforehand and the product evolved on an existing prototype. The duration of project VI was set at eight weeks and it took place between February and April 2005 (Table 19). A total of five iterations were conducted with 7.2 man months of software development effort for the project team.

The project team consisted of two experienced software developers from the case organisation and three software developers from the research organisation, who were members of the AR team of ENERGI. The latter were included to provide knowledge and experience on the adopted tools and practices of Mobile-D™. An in-house, on-site customer was available in the same premises yet not in the same open office-space and can, thus, be regarded as an off-site customer. The project team worked 24-hour weeks in an open-office space. The case organisation also included active management support and decision-making concerning the piloting, along with an external quality assurance team and an architect (Table 19). The ENERGI organisation provided the members of the support team (i.e., AR team) in the case project. The researcher was acting as an

external facilitator, who was responsible for conducting the SPI related activities and reporting the outputs iteratively to the management. The support team, as in ENERGI, was not available in the case organisation but an SEPG meeting was held at the end of the project. It comprised the developers participating in project VI, the management, and the on-site customer. The central goal of the group was to contribute to an organisation-specific agile software development process (paper VII). During the project, the facilitator provided the organisational management with iterative action point reports on how the project team was adapting their processes and why. This was done to enable the involvement of the management in the SPI activities during the project.

4.3 Collection of the Empirical Evidence

According to Yin (Yin 2003), "the use of multiple sources of evidence in case studies allows an investigator to address a broader range of historical, attitudinal, and behavioural issues" (Yin 2003, p. 98). Furthermore, the multiple sources of empirical evidence provide an opportunity for triangulation in order to make any finding or conclusion of the study more convincing and accurate (Yin 2003). Yin identifies six sources of evidence: documentation, archival records, interviews, direct-observation, participant-observation, and physical artefacts (Yin 2003). The multiple sources of evidence (Table 21) of this research include both qualitative and quantitative research data in various forms of documentation, archival records, interviews, and participant-observation.

During the research, 39 SPI related workshops were conducted within the project teams and for organisational purposes. This formed the main source of empirical evidence in this research consisting of, for example, experience notes from software developers and individual SPI actions (Table 21). Furthermore, the format as well as the content of the action point lists evolved throughout the research in line with the evolution of the applied SPI method. In Table 21, the number of data sources is defined as appropriate, regarding the resulting case documents.

Table 21. Collection of empirical material from the case projects.

Source of Evidence		Data type	Origin	Case Project					
				1	2	3	4	5	6
Documentation	Flap-sheets	Qualitative: personal experiences from previous iteration	Software Developers /PIW's	10	8	8	8	8	6
		Qualitative: Personal experiences and lessons learned from entire project	Software Developers /PIW's		2	2	2	2	4
	Project level SPI action point lists	Qualitative and quantitative: (Action points, responsibilities[2], follow-up[1], and qualitative validation[2]	Software Developers /PIW's	5	4[1]	4[2]	4	4	3
	Spread sheets	Quantitative validation data of SPI actions (analysed metrics)	Software Developers /PIW's				x	x	x
	Spread sheet[1]/ TaskMaster tool[2]	Quantitative: Time	Software Developers	x[1]	x[1]	x[2]	x[2]	x[2]	x[2]
	Spread sheets	Qualitative and quantitative: Defect data	Software Developers	x	x	x	x	x	
	SEPG action point lists	Qualitative and quantitative: improvement action points, responsibilities, follow-up, and qualitative validation during and between projects	SEPG meetings			1	6	4	0
		Qualitative: Feedback to project team	SEPG meetings			x	x	x	
Archival Records	Survey	Qualitative: Feedback on project level SPI activities	Software developers, customer, management						x
	Developers' diaries	Qualitative: Developers' notes	Software Developers	4	5	4	5		
	LOC counter	Quantitative: Logical size of end product	Software Developers	x	x	x	x	x	x
Interviews	Final interviews	Qualitative: Developers perceptions and experiences	Software Developers	x	x	x	x	x	x
	Scientific publication	Qualitative: Project team participation in Paper I	Software Developers	1					
Participant Observation	Field notes	Qualitative: participating project level SPI workshops	Researcher		1				
	Field notes	Qualitative: facilitating project level SPI workshops	Researcher		4	5	5	5	4
	Field notes	Qualitative: participating SEPG meetings	Researcher						1
	Field notes	Qualitative: facilitating SEPG meetings	Researcher			1	6	4	

In this research the data was collected from multiple sources using multiple data collection techniques including observation, collection of documentary evidence, focus group interviews (Morgan 1984, 1988, Patton 2002) and surveys. The multiple data sources were used within each of the case projects and the corresponding evidence was also collected from all the case projects of this research when possible, as illustrated in Table 21.

The SPI activities within the case projects as well as at organisational level provided qualitative output in the form of action point lists and flip chart -sheets with experience notes from the software developers, which were grouped by applying the KJ method (Scupin 1997). The first two project teams also collected time, size, and defect data in spread sheets, whereas the last four projects adopted a database tool (i.e., TaskMaster (Kyllönen 2005)) for the metrics data collection. The extensive data collection enabled quantitative validation of various SPI actions of the project teams and added the analysed metrics sheets, as well as the interpretations of the project teams in the selection of empirical research material. An LOC counter was used to systematically monitor the implementation of logical code lines in a quantitative manner. After each ENERGI project a final interview was held to record the perceptions and experiences of the software developers. The taped interviews were then transcribed for analysis. In project I, the project team also participated in the writing process of paper II, by collectively documenting their perceptions of the iterative team reflections that had been held. Furthermore, the author also took field notes throughout the case projects while participating in different SPI activities of the project team.

4.4 Storing and Analysing the Empirical Evidence

Yin (2003) addresses the lack of a formal database as a major shortcoming of most case studies and argues for its importance in constructing validity and reliability of the empirical evidence (Yin 2003). In the course of this study, a database was established in which the empirical material of the study was stored and made available for independent inspection, as suggested by Yin (2003). Figure 8 provides an overview of the structure, the contents, and the relationships between the data items stored in the database.

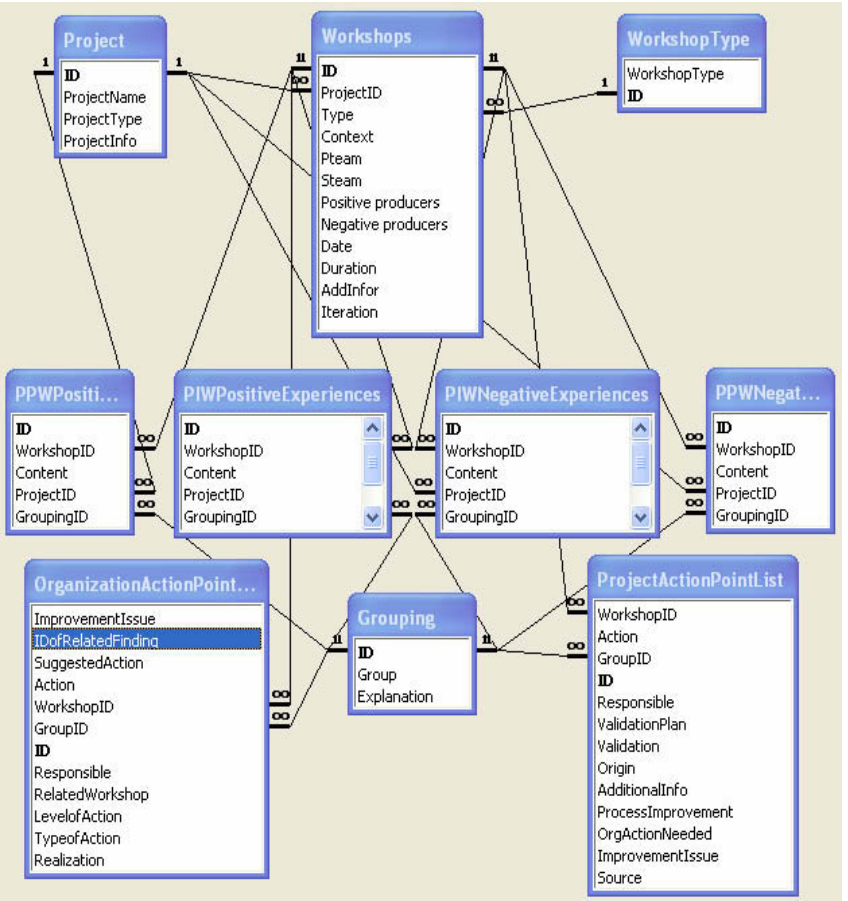

Figure 8. Empirical SPI evidence in Database Format.

The database supported a systematic classification of the individual experiences of the software developers and that of each of the resulting SPI actions in a manner that also would assure their linkage to each other and enable retrieval and analysis of the data in an effective manner. Thus, as suggested by Baskerville (1999), it was necessary to carry out a qualitative interpretation of the data through mapping, indexing, and scaling in order to enable any quantitative analysis. Other empirical evidence, such as metrics data collected and analysed in the case projects, interview material, field notes and survey results were stored in the same physical location in an appropriate format.

Following Yin's categorisation of strategies for analysing case study evidence (2003), this study falls into the category of developing a case description.

Although the objective of this research was not to be a descriptive one, the approach helped to identify the appropriate links and causalities in the empirical evidence, as suggested by Yin (2003). Furthermore, the descriptive insight provided in the empirical SPI material made it possible to tabulate the descriptive elements in a database form that also enabled its quantitative analysis (Yin 2003). The systematic transcription of the action point lists after each SPI session provided the researcher with an opportunity to simultaneously "play with the data", which is identified by Yin (2003) as a fruitful activity for generating a general strategy for data manipulation. As a result, the data items of SPI-related evidence and their relations were identified and it was realised that the extent, causality and the complexity of the available SPI-related research data would require effective analytic manipulation in some form or another to enable its management and analysis. The creation of a matrix placing the evidence within its categories, as suggested by Miles and Huberman (1994), was regarded as one way of manipulating the research evidence.

Yin's (2003) category of analysis techniques includes cross-case synthesis, which was applied throughout the multiple-case study of this research. Yin suggests that "cross-case synthesis can be performed whether the individual case studies have previously been conducted as independent research studies (authored by different persons) or as a predesigned part of the same study" (Yin 2003, p. 133) and if, at least, two cases are available for analysis. In connection with this study, the results of the analysis were published in scientific forums. Table 22 illustrates how the results of different publications were derived from both individual and cross-case analysis. As the study proceeded, it was possible to strengthen the findings by including an analysis of multiple case projects to draw cross-case conclusions on the current research focus.

Table 22. Focus of Papers.

Paper	Focus	Case Projects of Analysis
I	Research Context	-
II	Project Level SPI Activities	I
III	Project Level SPI Activities	I–II
IV	Integration of Project and Organisational SPI	I–IV
V	Project Level SPI method	I–V
VI	Integration of Project and Organisational SPI	I–V
VII	Deployment of Agile Methodologies	VI

4.5 Reporting the Results of Research

"Reporting a case study means bringing its results and findings to closure" (Yin 2003, p. 141). Yin (2003) suggests that some of the results should be composed early rather than waiting till the end of the data analysis process. In this research, seven scientific papers were published throughout the research process (Figure 9) to gain feedback continuously from the scientific community and to ensure the right orientation of the research. Thus, despite the written form of reporting the results, papers I, II, III, IV, and VII were also orally presented by the author in the corresponding scientific conferences.

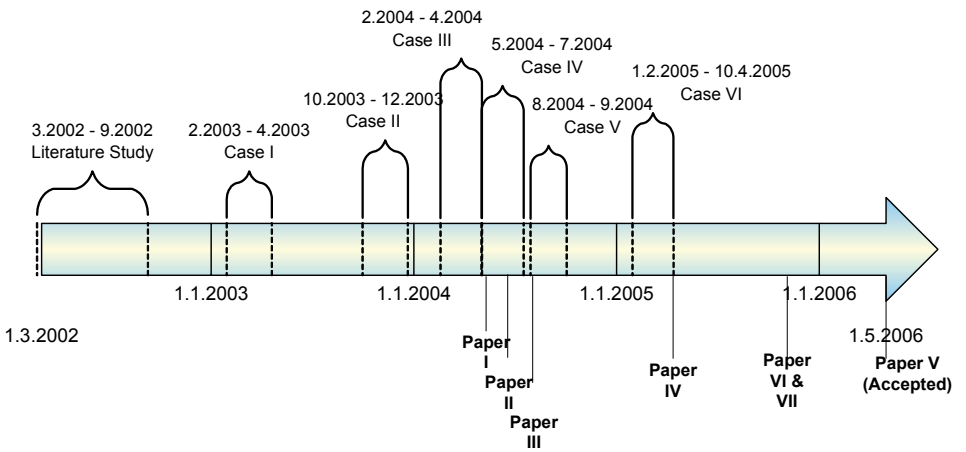

Figure 9. Publishing Process of the Research.

For all the papers, the publication forums have been carefully selected. The appropriate review process was one of the major requirements for the selection of the forums. Furthermore, the compatibility of the papers with the focus of the scientific publication forums was carefully considered prior to submitting each paper. One of the influencing factors was also the kind of audience that was regarded as important from the viewpoint of appropriate feedback. For instance, an early paper published on SPI in agile software development (paper I) was targeted at practitioners of agile software development (XP 2004), whereas Paper IV was submitted at LSO 2005, where experts of organisational SPI, addressed in the paper, were gathered.

Table 22 illustrates how the focus of the papers was defined on the basis of the increased understanding and readjusted focus of research. The seven papers of this research compose a continuum, where the results of the previous cycle of analysis are directing the focus of the research. In the following sub-sections, the focus and central conclusions of each paper are defined in more detail.

4.5.1 Paper I

Paper I introduces the context of the research that was applied in the first five case projects of this thesis, i.e., ENERGI. In the paper, a new concept of studying agile software development in close-to-industrial settings is suggested. From the viewpoint of this research, it was considered very important to gain acceptance from the scientific community in the form of positive feedback for applying such a research approach.

In paper I, there is a discussion on how the proposed research approach is constructed on the strengths of both the case study and experimentation research approaches. The paper provides a context for conducting research-oriented software development in a way that will benefit both research and industry.

The author of this thesis is the principal author of the paper and is the main contributor in documenting the novel research context that was evolved and applied at VTT. Professor Pekka Abrahamsson has had a central role in innovating and establishing the ENERGI research context. The ENERGI context is further discussed in section 4.2.1.

4.5.2 Paper II

In paper II, the focus is on the project-level process-adaptation mechanisms within agile software development teams. The researcher is the first and main author of the paper. The project team members, however, have contributed to the paper by writing their perceptions and, naturally, by participating in the SPI activities throughout the project. In Paper II, the research data from case project I was analysed in order to provide empirical evidence on process adaptation within the agile project team, by combining elements from both Cockburn's

team reflection workshop (2002) and the lightweight postmortem review technique suggested by Dingsøyr and Hanssen (2002).

From the viewpoint of the SPI method used, one central finding of the analysis was that the process adaptation method did not provide a means by which to evaluate the effects of the adaptation on, for example, the efficiency of the development or quality of the product. The major deficiency identified was the lack of follow-up and validation mechanisms in the current process adaptation methods. Essentially, validation has been identified as one of the critical success factors of SPI (Komi-Sirviö 2004). Another important finding of the study was that the adopted techniques as well as the SPI ideologies of agile software development failed to address the learning of project teams across projects rather than merely focusing on improving the performance within individual project teams.

In general, one central finding in Paper II was the positive attitude and willingness of the software developers to take part in the process adaptation activities. This was regarded as an important finding, as both the lack of participation and commitment can be seen as major failure factors in traditional SPI initiatives (Abrahamsson 2002, Komi-Sirviö 2004). The research data from case project I implies that such positive perceptions are caused by the immediate visibility of the process improvements for the developers as well as by the possibility to influence the daily working practices in a specific way.

4.5.3 Paper III

The focus of paper III is on the project-level process-adaptation activities among the agile software development teams. In paper III, the author is the main and sole contributor.

In paper III, the conclusions and empirical research data from the first case project are compared with case project II in order to find correlations and deviations between the findings. The quantitative and qualitative data again suggest that, among agile software development project teams, process adaptation is regarded by the developers as a specific way to improve and adapt agile software processes during the iterative cycles of software development.

Thus, the findings of Paper III suggest that process adaptation should be regarded as a useful part of agile software development projects, especially if the existing methods are supplemented by follow-up and validation of process improvements. The findings in case project III reinforce the findings of case project II, regarding the positive effects of iterative SPI workshops on the satisfaction and learning of software developers in their daily working practices. However, in Paper III it became clear that measurement data for validating the process improvements should be considered to provide a means to evaluate the effects of the change, as suggested in (Beck 1999). It is also suggested that the existing techniques fail to address the integration of the extensive learning of the project teams in organisational SPI activities.

4.5.4 Paper IV

Based on the findings of case projects I and II reported in papers II and III, paper IV focuses on defining mechanisms to integrate agile adaptation of project teams in a larger organisational SPI context. Accordingly, the focus of case projects III, IV, and V increasingly aim to provide solutions for such collaboration between agile software development teams and organisational SPI stakeholders.

Paper IV, despite being a short workshop paper, can be regarded as an important publication, where the central issues for improving the existing iterative SPI workshop techniques are suggested, as they were encountered and implemented in the first four case projects of the research. The means for a systematic follow-up and validation of process improvements as well as a structured documentation on relevant information of SPI actions and their results, are proposed in order to link the team-centred SPI within agile software development with the concepts of organisational learning and organisational SPI.

In paper IV, the author is the main and sole contributor. In consequence of the paper, the author was also invited to give a panel statement and to participate in the panel discussion at the LSO 2005 to discuss the topic of "Spreading software engineering experience through communities of practice and experience packaging". This was an important forum for the researcher in terms of gaining direct feedback on the results of the research from the specialists of the field.

4.5.5 Paper V

In Paper V, which was a journal paper, an Iterative Improvement Process is proposed. The paper includes the empirical evidence and analysis of the first five case projects. This paper represents the culmination of the project level SPI focus of this study. While the Iterative Improvement Process aims at providing an SPI method to be used in an iterative improvement and adaptation of individual agile software development project teams, it also aims at providing clear links to enable the integration of project level improvement activities and organisational SPI. In addition, the evidence from the case projects indicate the capability and willingness of agile software development teams to improve their development processes with small and simple, yet effective and visible improvements during the projects. However, it is also proposed that systematic SPI mechanisms and organisational support are of central importance in iterative improvement in agile software development teams.

4.5.6 Paper VI

In Paper VI, as in paper IV, the focus is on providing a tentative model for integrating the adaptation activities of agile software development and the traditional activities of organisational SPI stakeholders to enable a mutually beneficial existence of the two. In Paper VI, the empirical results from a longitudinal case study over five software development projects (case projects I–V) are presented to support the integration of agile software development and organisational SPI. The study reveals the great importance of close collaboration between the organisational and project levels throughout the project. The suggested SPI model identifies the organisational and project level SPI activities, the relevant stakeholders, and identifies and defines the activities needed to enhance SPI within agile projects and at an organisational level, these activities include constant collaboration, support, continuous monitoring and feedback, facilitation, and packaging of process knowledge.

In paper VI the researcher in this thesis is the first author and the main contributor.

4.5.7 Paper VII

In paper VII of this thesis, a framework for deploying agile practices and processes in software development organisations is proposed. The framework was designed as it became evident that software development organisations were increasingly interested in adopting agile processes and practices, while lacking procedures and methods for supporting a systematic selection and deployment of new agile practices, and tailoring them to suit the organisational context. In the paper, the central results of the earlier research of this thesis (such as the Iterative Improvement Process) are integrated in the process of deploying agile practices. Thus, in paper VII, the application of the iterative improvement in an industrial context (i.e., case project VI) is reported. It illustrates how the method can be used in an organisation when adopting and adapting agile practices and providing validated feedback for agile assessments.

The author of this thesis is the first author of paper VII jointly with Mrs. Minna Pikkarainen. The two researchers designed and reported the agile deployment framework in close co-operation and equally sharing the main contribution to the paper. The parts of the paper which deal with agile assessments were mainly provided by Minna Pikkarainen, whereas the author of this thesis has been responsible for designing and reporting the Iterative Improvement Process related issues in the paper. The author of this thesis has also conducted the iterative improvement sessions (i.e., PIW's) and the related research in the case organisation. Mr. Jari Still has been managing the SPI activities in the case organisation and, thus, has had an important role in conducting different agile deployment activities in the case organisation.

4.6 Summary

The research is based on empirical evidence from six case projects that were conducted within a period of over two years, between February 2003 and April 2005. In addition, a literature review was conducted before the case study sessions to provide a theoretical basis for the research. Various research methods were used during the study and seven main stages of the research can be identified within the evolution of this research: the literature review, the case study (case project I), and five cycles of AR (case projects II–VI).

The different stages form a continuum in which the focus of research was continuously adjusted based on the results of the previous stages. The first two case projects, and, consequently, papers II and III, and V, focus on process adaptation activities among agile software development teams. From project III onwards, the research also included the organisational SPI aspect and its integration into the process adaptation activities of software development teams. The results were published in Papers IV and VI. In Paper I, the context of the research in case projects I–V is defined. In addition, Paper VII proposes an agile deployment framework and suggests how the process adaptation method built during this research can be implemented as a useful part in deploying agile software development methodologies in software organisations.

5. Evaluation of the Research

In this section, the different issues addressed by this research are evaluated. First, in sub-section 5.1, describes how the selected research methods were applied in the research. In sub-section 5.2, the results of the research are evaluated from the viewpoints of theory (5.2.1) and practice (5.2.2).

5.1 Validity of Research

In all the six case projects of this research, the case study method was applied. In addition, the last five case projects were conducted as AR case studies. The quality of any empirical social research, including case studies, can be evaluated by four tests: construct validity, internal validity, external validity, and reliability (Yin 2003). Furthermore, Yin has suggested (2003) various tactics for ”dealing with these four tests when doing case studies” (Yin 2003, p. 34). In Table 23, the four tests and the tactics recommended by Yin are presented in grey sections. The white sections of Table 23 summarise how each of the recommended tests and tactics were accomplished in the case projects of this study and in which part of the thesis the tactics are reported in more detail.

In this research, the construct of validity was ensured by multiple sources of evidence including four of six identified data sources by Yin (2003): documentation, archival records, interviews and participant observations (see Figure 8). The construct of validity was further ensured by submitting reports of case studies continuously throughout the case projects (Papers I–VII) to be reviewed and published in various, carefully selected publication forums (4.5). The papers included citations of various evidentiary sources of the case study database to maintain the chain of evidence. The reports were targeted at a diverse set of audiences, including academic colleagues and practitioners, as suggested by Yin (2003). The ongoing publication process ensured for the researcher continuous feedback on the validity and usefulness of the research. In addition to the case study reports, the case study database also included an evidentiary base (Yin 2003) where a quantity of empirical evidence was systematically stored in tabular format for retrieval and analysis (Figure 8). The evidence in documentary form was stored in the same location.

Table 23. Case Study Tactics in the Case Projects.

Test	Case Study Tactic	Implementation	Definition
Construct Validity	Use multiple sources of evidence	Documentation Archival records Interviews Participant-observations	Section 4.3 Table 21
	Establish chain of evidence	Case study reports (Papers I–VII) Creation evidentiary database	Section 4.5 Section 4.4, Figure 8
	Have key informants review draft case study report	Publication process of case study reports (Papers I–VII)	Section 4.5, Figure 9
Internal Validity	Pattern-matching Explanation-building Addressing rival explanations Logic models	Pattern-matching Explanation-building Addressing rival explanations	Papers II–VII
External Validity	Use theory in single-case studies	Series of case studies (VI case projects)	Section 4.3
	Use replication logic in multiple-case studies		
Reliability	Use case study protocol	Data collection methods, procedures	Section 4.3, Table 21
	Develop case study database	Sources of evidence Evidentiary database Case study reports (Papers I–VII)	Section 4.3, Table 21 Section 4.3, Table 21 Section 4.4, Figure 8 Section 4.5

Yin suggests pattern-matching as one of the most desirable techniques for case study analysis (2003). In this study, the pattern-matching was done by exploring, for example, regular trends among the number of iteratively generated findings of software developers on the weaknesses and strengths of the software process and the number of generated SPI actions. This pattern-matching enabled further evaluation of the identified patterns in order to seek explanations among rival explanations – between and within the case projects. An example of this was the attempt to find and verify different explanations behind the downward trend in positive findings among the software developers, while the downward trend in corresponding negative findings strongly implied iterative improvement of the software process.

Due to the explanatory nature of the case studies of this research, explanation building was found to be a relevant tactic for ensuring the internal validity of the research (Yin 2003). In the publications of this research a narrative form of explanation building supported by quantitative evidence was used to illustrate the chain of evidence of why, how and with what effects the project teams used quantitative validation of SPI actions, i.e. if the quantitative validation actually could be demonstrated to provide added value for the SPI among project teams (Paper V).

According to Yin, external validity refers to the problematic issue of generalising the results of case studies (2003). Case studies rely on analytical generalisation where "the investigator is striving to generalise a particular set of results to some broader theory" (Yin 2003, p. 37). Yin suggests that the replication of case studies in a context "where the theory has specified that the same results should occur" (2003, p. 37) provides a basis for accepting the results provide strong support for the theory. In this research, increased versions of SPI activities in the agile software development context were conducted in six consecutive case projects. This provided an opportunity to evaluate whether or not the SPI mechanisms actually seemed to have parallel effects in different project teams. The first five case projects were conducted in fairly similar, close-to-industrial research contexts (Paper I), where various independent variables remained unchanged. This provided an opportunity to validate the research from the viewpoint of external validity. The last case study, which was conducted in a fully industrial context, also provided an opportunity to evaluate to what extent the results could be considered generalisable. Naturally, the ENERGI case projects provide (case I–V) a limited and novel context of research and, thus, limit the opportunities for drawing broad generalisations. The implementation of the results of this research in diverse industrial settings would highly increase the validity and generalisability of the results.

Yin suggests enforcing the reliability of research "to minimize the errors and biases in a study" (2003, p. 37). A reliable case study is defined as one that can be conducted again by another investigator with the same findings and conclusions. Yin further notes that "the emphasis is on doing the same case over again, not on "replication" the results of one case by doing another case study" (Yin 2003, p. 37). The tactics suggested for ensuring the reliability of a case study are to use the case study protocol and to develop a case-study database. A

case-study protocol is especially essential when conducting a multiple-case study, as in this case. Furthermore, it has been defined to contain the instruments, procedures, and general rules to be followed along the research including an overview of the case projects, field procedures, case study questions, and a guide for reporting the case study (Yin 2003). In addition, the researcher's role as a facilitator and her active involvement with expected benefit for both research and practice – as suggested to be the case in AR (Baskerville 1999) – can be considered as a factor of bias in this research.

In the line of this research, a preliminary case study protocol was established and incrementally complemented along the research. A proportion of the protocol could be defined up-front and systematically followed throughout the multiple-case study including, for example, final interviews and researcher diaries as qualitative data collection mechanisms. However, due to the adoption of the AR method from the second case project onwards, and the close link between the research data collection and the SPI methods under investigation, it was also necessary to continuously modify certain aspects of the case-study protocol throughout the study. For example, the database tool for metrics collection (Table 21) and the structured template for storing SPI action points among project and SEPG teams (Table 18) were defined during the research project as a result of the research itself. However, even though new data items were added to the sources of evidence and the data collection and storage procedures were enhanced and adopted along the research, so as to make sure that all the preliminarily defined data items were still collected within all the projects of the multiple-case study. In addition, the systematic storing of empirical evidence was planned and conducted by reporting the results of the study as well as establishing an evidentiary database for the research data.

5.2 Evaluation of the Results

In the research four specific outputs can be identified:

1) a Controlled Case Study approach for integrating agile software development research and production (Paper I),
2) a project level method for conducting process adaptation activities, i.e., Iterative Improvement Process (Papers II, III, and V),

3) a tentative model of integrating the traditional SPI activities of organisational stakeholders and the Iterative Improvement Process activities of software development teams (Papers IV and VI), and
4) an agile deployment framework in which the Iterative Improvement Process is a central feedback mechanism (Paper VII).

In this section, the outputs of this research are evaluated from the viewpoints of theory and practice.

5.2.1 Implications for the Theory

From a theoretical viewpoint, the aim of this study was to increase and extend the body of SPI knowledge in the area of agile software development concerning the following issues:

1) how to conduct SPI in individual agile software development teams, and
2) how to integrate the agile SPI activities of individual project teams and traditional continuous organisational SPI activities.

Firstly, the aim of the researcher was to apply AR in order to construct an SPI method suitable for individual agile software development teams. During the research, it became evident, that the gaps in the existing agile SPI methods often concerned aspects considered as critical in the context of traditional SPI. However, it was evident that, in order to maintain the fundamentals of agile software development, the traditional SPI mechanisms would not apply as such. The Iterative Improvement Process (Paper V) embodies the findings and learning from the SPI activities conducted together with the researcher and the six case project teams. It can be considered as the main outcome of the project level SPI research of this study and proposes how to conduct SPI in individual agile software development teams. In the SPI process, however, the integration points between continuous organisational learning and iterative reflection of agile software development teams are identified (Paper V).

The starting point for the proposed SPI process was in the existing agile SPI methods, especially the Reflection workshop technique (Cockburn 2002, 2005)

and Postmortem review technique (Dingsøyr & Hanssen 2002). The resulting SPI process aims at identifying the improvements in an iterative manner by providing systematic yet rapid procedures, to further defining and documenting the resulting actions and to follow-up with qualitative and quantitative validation of their implementation and success. From the viewpoint of traditional SPI with an organisational improvement focus, this provides an opportunity to gain validated process improvement knowledge from agile software development teams.

Various problems have been encountered in the context of traditional SPI regarding, for example, the low commitment to SPI activities (Abrahamsson 2002), their actual effectiveness in improving the software development practices of organisations, the volume of effort needed for implementing the SPI initiatives and the low speed at which visible and concrete results are achieved (Dybå 2000, Goldensen & Herbsleb 1995, Krasner 1999). In fact, it has been reported that approximately two-thirds of traditional SPI initiatives fail to achieve the intended goals (Debou 1999). Abrahamsson (2001) proposes that through voluntary involvement and by embedding SPI in the daily routines of software engineering, the commitment towards SPI activities would increase. The empirical evidence of this research implicates that the iterative process adaptation of agile software development teams provides opportunities to respond to the core problems of traditional SPI. For one, both the qualitative and quantitative data of this research indicate that the agile teams were both capable and willing to improve their development processes by participating in the activities of the Iterative Improvement Process. Such a positive attitude on the part of developers to participate in SPI activities was, according to the developers, caused by having an opportunity to actually affect the daily working practices, the fact that the agreed improvements were actually implemented and supported, as well as the visibility and rapid execution of the improvements. With 1.9% of the total effort of software development, the five project teams identified, planned and agreed a total of 182 improvements. Thus, for one, it can be argued that this research supports the suggestion of Abrahamsson (Abrahamsson 2002) that voluntary involvement and embodying SPI as a part of software engineering practices will have a positive effect on the commitment of software professionals towards SPI. Respectively, the research results indicate that the agile approach to SPI, i.e., the regular reflection of software development teams, will also have positive effects on the speed and visibility of SPI, especially from the viewpoint of software developers. However, the

research data also indicates that without organisational participation and support, as much as 33% of the planned improvement actions will fail. Thus, the empirical evidence of this research also indicates that one critical success factor of Iterative Improvement Process is the participation and support of organisational stakeholders rather than leaving the individual software development teams to reflect and adapt the process by themselves.

The second focus of this research was to study how the agile SPI activities of individual project teams and traditional continuous organisational SPI activities could be integrated in a mutuallybeneficial manner. It has, indeed, been claimed that there is a need to extend agile methodologies and adapt them to organisations with established and mature plan-driven processes (e.g., Boehm & Turner 2005). In addition, one major issue in adopting agile methodologies in organisations is the problem of balancing the currently dominating engineering ideologies and methodologies of manageable, predictable and repeatable processes with the agile software development methods, which again embrace self-organisation, process adaptation and constant changes (Lycett et al. 2003). In this study, it is suggested that the reflection activity (i.e., the Iterative Improvement Process) of agile software development teams can co-exist with traditional continuous organisational improvement activities (Papers IV and VI). During the research it was revealed that while the development teams external support and participation, the organisational stakeholders in turn benefit from the iterative process improvement knowledge of software development teams. The mechanisms for bidirectional knowledge transfer (Paper IV) and mutually beneficial collaboration (Paper VI), however, needed to be put into place. In addition, the organisational aspect of this research results in an agile deployment framework, which incorporates Iterative Improvement Process as a rapid feedback mechanism in a traditional QIP cycle and, thus, provides an integration point between software development teams and organisational SPI stakeholders (Paper VII).

Komi-Sirviö (2004) has studied a wide range of SPI literature in order to identify the critical success factors (CSF) of SPI methods. In Table 24, this CSF framework is used to evaluate the theoretical implications of the central results of this research. The goal is to examine how well the proposed SPI process for agile software development teams corresponds to the factors proposed as critical for any SPI method.

Table 24. Evaluating the Results of the Research Using CSFs.

CSF Framework			Research Results	
Main Class	CSF	CSF Evaluation	Definition	Reference
Improvement Management	1	Does the method support different SPI approaches?	Yes. The Iterative Improvement Process activities are integrated into the continuous improvement cycle of QIP.	Papers IV, VI
			Yes. Various methods and techniques are suggested in Iterative Improvement Process activities to support the process adaptation among software development teams.	Papers II–VI
			Yes. Organisational guidelines and facilitation are suggested to be used in the Iterative Improvement Process.	Papers V, VI
			Yes. The Iterative Improvement Process is integrated as part of the organisational process deployment framework.	Paper VII
	2	Does the method support the participation of all affected parties?	Yes. SPI participants and activities of both organisational and project levels are identified: software development team, facilitator, and SEPG, and management.	Papers II–VII
	3	Does the method support co-operation with software engineers?	Yes. The method of Iterative Improvement Process enhances the control of software developers in SPI activities.	Papers II–VII
			Yes. The iterative and continuous co-operation activities between software developers and SEPG are addressed and defined.	Paper VI
			Yes. The continuous co-operation between management and software developers is addressed.	Paper VII
	4	Does the method support planning and carrying out training as part of the initiative?	Yes. It is suggested an SPI facilitator assists software development teams in the Iterative Improvement Process.	Papers II–VII
Commitment	5	Does the method support the commitment of top managers?	Yes, partially. Mechanisms for providing SPI knowledge and improvement opportunities from projects to organisational utilisation have been established.	Papers IV, VI, VII
	6	Does the method support the commitment of middle managers?	Partially. The active participation of organisational stakeholders has been addressed and defined.	

CSF Framework			Research Results	
Main Class	CSF	CSF Evaluation	Definition	Reference
	7	Does the method support commitment of software engineers?	Yes. The Iterative Improvement Process identifies the software developers as a central origin of SPI actions.	Papers II–VII
			Yes. The method of Iterative Improvement Process addresses the rapid and visible implementation of SPI activities.	
			Yes. Mechanisms are established to provide software developers with iterative SPI feedback from organisational stakeholders.	Paper VI
			Yes. Mechanisms are established to provide software development with continuous support in their Iterative Improvement activities.	Papers IV, VI
Culture	8	Does the method support developing improved solutions on a case-to-case basis?	Yes. The process adaptation among Iterative Improvement Process occurs in individual project teams and, thus is context specific and based on the experiences and knowledge of software developers.	Papers II–VII
Plan	9	Does the method support clarifying the current status of processes?	Yes. The Iterative Improvement Process is identified as a feedback mechanism for organisational SPI, e.g., process assessments.	Paper VII
			Yes. The software development teams iteratively assess the weaknesses and strengths of their current software processes.	Papers II–VII
	10	Does the method support establishing a link between business and improvement goals?	Yes, partially. The defined co-operation between organisational SEPG and software development teams and the role of the facilitator enable establishing a link between project and organisational SPI goals.	Papers II–VII
	11	Does the method support measurable improvement goals?	Yes. The action point template is proposed to define the SPI actions and to plan means for their validation.	Papers IV, V, VI
	12	Does the method support generating an improvement plan?	Yes. The action point template is proposed to support the systematic and structured generation of improvement plan including the definition of tasks, schedule, resources, reporting, follow-up, and validation.	Papers IV, V, VI

CSF Framework			Research Results	
Main Class	CSF	CSF Evaluation	Definition	Reference
Do	13	Does the method support the testing of developed solutions in a pilot project?	Yes. From an organisational viewpoint, the Iterative Improvement Process is integrated into the deployment framework which supports piloting.	Paper VII
			Yes. From a project level viewpoint, process improvements are systematically tested and validated during iterative cycles of software development.	Papers II–VII
Check	14	Does the method support using metrics in monitoring improvement actions and results?	Yes. From an organisational viewpoint, the monitoring activities of SEPG team are identified and defined.	Paper VI
			Yes. From a project level viewpoint, the metrics have been defined as a tool for iteratively monitoring and validating the improvement actions.	Paper V
Act	15	Does the method support the sustainability of an improvement initiative?	Yes. The storing and transferring of project level SPI knowledge to organisational level support the organ-isational sustainability of improvements.	Papers IV, VI
			Yes. The agile specific process deployment framework supports the sustainability of improvement initiatives.	Paper VII

Table 24 illustrates how the CSFs of SPI are, in various ways, embedded in the various SPI outputs of this study.

In addition, one implication for the theory of this research can be the controlled case study approach proposed and applied in this research (Paper I). The approach suggests how to integrate industrial agile software development and research in order to generate impact on both the scientific and practical software engineering community. The controlled case study approach has been applied in the first five case projects of this research conducted in ENERGI context.

5.2.2 Implications for the Practice

As a characteristic of any AR study, this research provided results on the practical implications during the case projects as well as more general implications for software engineering. In the following, the two aspects of the practical implications from this study are discussed.

During the six case projects of the study, the software development teams agreed to perform a total of 182 actions to improve their daily work. These included: 1) actions directly related to SPI issues such as the working procedures and tools of software developers (44%), and 2) actions related to other concerns of developers such as acquisitions concerning the working environment (56%). The developers spent 1.9 of the total effort in participating in the Iterative Improvement activities, i.e. identifying, planning, agreeing and evaluating the effects of the process improvements. On the other hand, it was also revealed that external (i.e., organisational) support was also needed in the implementation of 33% of the improvement actions. Furthermore, it was also revealed, that without a systematic means to define, document and follow-up the project level SPI activities, as much as 35% of the agreed action points would have remained incomplete.

On the one hand, the numerous improvement actions (average of 30 per case project, approximately from seven to eight improvements per PIW) and the reasonable amount of effort needed for SPI activities indicate that the project teams were able to identify minor, yet effective, actions to improve their daily work. The effectiveness of the SPI actions was detected also in the positive attitude of software developers' participation in the SPI activities (i.e., PIW's). The possibility to affect the daily working practices as well as the rapid application and visibility of the improvements were considered important factors for the success of Iterative Improvement Process by the software developers. This also indicates the importance of actually implementing the identified improvements and, thus, for the need for organisational support and systematic mechanisms of SPI. On the other hand, a considerable number of identified improvements of the software development teams related to more daily concerns rather than process issues. This indicates that Iterative Improvement and PIW's can also serve as ordinary project meetings, at which the state of the project and the everyday needs and concerns of the project team can be addressed and acted

upon. Papers II, III and V focus on project level SPI in agile software development teams.

At the organisational level, a total of 112 improvement actions were taken by the AR team functioning in the role of SEPG in ENERGI. These included more longterm changes to the organisational software development-related tools and process (i.e., Mobile-D™), as well as the activities related to support the development teams in their SPI activities. To a large extent, the improvement activities of the SEPG team were based on the process knowledge emerging from the software development teams, i.e., their requests, findings and inventions. However, during the research it became evident that the traditional activities of the SEPG team would need to be altered to fit the context of agile software development. While the agile software development teams were found to need continuous and iterative support in their SPI activities the SEPG team was found to largely benefit from the iterative SPI knowledge of the agile software development teams. Thus, mechanisms were needed to enable the mutually beneficial co-existence of the two SPI stakeholder groups in the organisation. Papers IV, VI, and VII focus on the organisational aspects of SPI in the context of agile software development.

As a result of the AR conducted iteratively during the case projects, three main implications for the practice of software engineering can be identified:

1) an Iterative Improvement Process for agile software development teams,
2) tentative mechanisms for integrating the SPI activities of individual agile software development teams and traditional organisational SPI actors (Papers IV and VI), and
3) an agile specific deployment approach for organisations adopting agile software development technologies (Paper VII).

Firstly, software development teams are provided with a practical process adaptation and improvement mechanism called “an Iterative Improvement Process”. It provides mechanisms consistent with the fundamentals of agile software development, yet still embodies the vital elements identified as critical for any SPI method. However, the Iterative Improvement Process also aims at providing integration points to link project level SPI activities with

organisational level SPI. The SPI process itself with empirical evidence is presented in Paper V while Papers II and III focus on presenting the empirical research data revealing, for example, the surprisingly positive attitudes of software developers towards the opportunity of rapidly tackling the weaknesses and problems in their working process. The proposed Iterative Improvement Process for agile software development teams can be considered to be applicable in such software development teams where the software process is incremental or in whose processes clear intermediate points exist. The software development team should also be able to regularly gather together to discuss relevant issues in a face-to-face manner. However, even though the suggested Iterative Improvement Process was developed among small development teams applying agile software development methodologies, neither of these aspects is required to conduct SPI as suggested in this study.

Secondly, the research uncovers practical implications for software development organisations that value continuous organisational learning while also applying agile software development technologies. Based on the research, it is proposed that the traditional activities of organizational SPI may need to be altered to enable the mutually beneficial and effective co-existence of agile projects and organisational SPI. Mechanisms for constant collaboration, organisational support, continuous feedback, knowledge transfer and facilitation are proposed and discussed in Papers IV and VI. From the organisational viewpoint, the suggested mechanisms enable the integration of SPI in agile software development teams and organisational SPI stakeholders, and are especially applicable if the organisation addresses continuous organisational SPI and plans to conduct agile software development projects or similar iterative process models. Another requirement is that the organisation must appreciate and highly value knowledge and learning among its software development teams and should be willing, to some extent, to pass the control of SPI from organisational to project level.

Thirdly, the research results offer practical implications for software development organisations deploying agile software technologies. From the viewpoint of this research, the main implication for practice is how the project level Iterative Improvement Process and its feedback mechanisms can provide benefits to organisations deploying agile software development technologies. In this respect, in the agile deployment framework, the iterative process adaptation

and improvement of agile software development teams is integrated as part of the organisational deployment process. An example of the agile deployment framework is presented in Paper VII.

Finally, it should be noted that in applying the methods and suggestions made and based on this research, the organisational context is always in a central role. Thus, the methods and activities always need to be adapted with respect to the existing (SPI) practices and culture of the organisation.

6. Summary and Conclusions

In this section, the results of the research are summarised (6.1), and the limitations (6.2) and further research avenues (6.3) are discussed.

6.1 Summary of the Results

In this research, two main research questions were formulated, which are set out below with a summary of the results.

Q.1. The first research question is 'how to conduct SPI in individual agile software development teams?'

As a result of the collaboration of the researcher in the six agile case projects, an Iterative Improvement Process (Paper V) was defined and suggested for implementation in SPI among agile project teams in an agile specific manner. In section 5.2, there is a discussion on how the critical success factors of SPI (Komi-Sirviö 2004) are embedded in the proposed SPI method.

The role of commitment has been found to be significant for the success of any SPI initiative (Abrahamsson 2002). In addition, team satisfaction has been identified by Koch (2005) as one central aspect in evaluating the effects of an agile method. Thus, in this thesis these aspects were examined to discover how the software developers perceive the effects of process adaptation in their daily work and the level of iterative participation in improvement. Papers II and III present empirical evidence from case projects which indicate highly positive attitudes on the part of software developers towards being able to address and improve, rapidly, the weaknesses and problems in their daily working practices. However, during the research (Papers V, VI, VII), it was also observed that, without external support, a significant part of the SPI actions by the project teams could not have been implemented.

The second research question, addresses the organisational level focus of the research, and is:

Q.2. How to integrate the agile SPI activities of individual project teams with traditional organisational SPI activities?

In this thesis, the research included the related aspects of continuous organisational improvement and the deployment of agile methodologies in organisations.

Originating from the traditional SPI context, one notion behind the research was that the process adaptation activities of agile project teams would provide valuable knowledge also for organisational SPI purposes (Paper IV). During the research, it was discovered that the case project teams also needed support from other stakeholders to implement a major part of their SPI activities (Paper V, VI, and VII).

Thus, based on the empirical results of Papers IV and VI, mechanisms were defined as to how and what process knowledge could be transferred from project teams to organisational SPI stakeholders (Papers IV, VI) and how the traditional SEPG team should adapt its activities to suit the SPI mechanisms of the agile software development teams (Paper VI). Collaboration activities between the agile software development teams and the SEPG team were identified as necessary to support the co-existence of traditional, yet agile, adapted SPI mechanisms at organisational level and agile software development and SPI in agile project teams. The resulting output is a tentative model for integrating agile software development with organisational SPI.

The final case project focused partially on deploying agile methodologies in industrial organisations. As a result, an agile deployment framework is proposed (Paper VII) whereby the Iterative Improvement Process is integrated into organisational improvement as a feedback mechanism for agile assessments (Pikkarainen & Passoja 2005) and traditional organisational SPI models.

6.2 Limitations of the Study

In this research, several limitations can be identified. For example, the first case projects of the research were conducted in a close-to-industrial setting, rather than in a “pure” industrial environment. In addition, the case project teams included

information processing science students as software developers. These two factors can be considered as limitations with regard to the generalisability and validity of the results of the research. The problem of generalisation, however, is present in any case study where the results only present the truth in a given context and the exact replication of the study is not possible. On the other hand, the ENERGI research context (Paper I) provided an optimal opportunity – both for the researchers as well as the customer organisations – to intensively examine, empirically evaluate, learn and further develop agile methodologies at a very early stage and prior to their deployment in an industrial environment, while implementing real software products for customers. Moreover, the deployment of the SPI mechanisms subsequently in a pure industrial environment (Paper VII) corroborated, in many respects, the same empirical results as those from the case projects within the ENERGI project (Papers II–VI).

Furthermore, the researcher's role as a facilitator can be considered as a factor of bias in this research. However, the adoption of AR was considered as a suitable and a highly practical method of research in the rapidly changing environment of agile software development. For the researcher, it provided an opportunity to utilise her prior SPI knowledge, yet to continuously learn from the experience and knowledge of software developers in the development of SPI mechanisms. From the viewpoint of the software developers, the continuous SPI support and collaboration with the researcher were of great benefit in the task of solving the real daily problems in the development process.

The organisational SPI mechanisms (i.e., SEPG activities) of case projects III-V were established for the purposes of the research organisation and the ENERGI context, rather than to be applied within the customer software development organisations themselves. Thus, the organisational SPI focus was to systematically and continuously improve the Mobile-D™ (Abrahamsson et al. 2004, Ihme & Abrahamsson 2005) agile software development methodology, which was adopted by the case project teams. In this research, Mobile-D™ was used as an organisational software development process of ENERGI separate from the still traditional software development processes of the customer organisations.

The separate SEPG, which was independent from the customer organisations and consisted of the AR team, may be considered as an artificial stakeholder

group which can be used in the study of social contexts, as suggested in AR. Thus, while the SEPG team can be claimed to only simulate the 'real' industrial environment the established SEPG team had still real focus of SPI. For one, from the customer viewpoint, the SEPG activities aimed to increase the effectiveness of the software development and the quality of the end product in the case project teams and from case to case. Secondly, the SEPG activities defined a process model (Mobile-D™) to be adopted in software development organisations (e.g., in case VI). In addition, from the viewpoint of software development teams, the SEPG activities provided constant support in their software development and SPI activities. The ENERGI context also provided an early opportunity to study the organisational SPI mechanisms in the context of agile software development which, at the time of the study, was not possible in industrial organisations still focusing on deploying agile methodologies in single project teams. The resulting tentative model of SEPG activities in agile context, however, still would need to be further evaluated in diverse industrial contexts.

In the ENERGI context, the project teams were also working in a more insular environment, where the interaction with other projects and organisational stakeholders or the overlapping schedules and tasks were not necessarily present to the extent of industrial environment. In the ENERGI research context, the possibilities for controlling or limiting the influencing factors that might affect the results of the case studies can be regarded as positive or negative, depending on the viewpoint.

6.3 Future Research

On the subject of SPI in the context of agile software development, there are several avenues of investigation to be explored in future research. Firstly, the importance of storage, retrieval, management and analysis of SPI knowledge from previous projects has been widely discussed among traditional SPI methods (e.g., Basili & Caldiera 1994, 1995, Fitzgerald et al. 2004). However, it would be valuable to further study how the use of process improvement knowledge, emerging from agile software development teams, may also benefit organisational learning in different contexts. The future research on agile software development and SPI should include an examination of how rapidly accumulating process knowledge of agile software development teams could be

effectively managed to facilitate cost-effective organisational learning (Garvin 1993, 2000) in a bottom-up manner. Thus, for example, tool-supported knowledge management (KM) (Nonaka & Hirotaka 1995) mechanisms should be studied in an agile software development context. In particular, studies should focus on how and what kind of tools could support knowledge transfer from individual projects to organisational level while being used by project teams as well. In addition, the tentative model of integrating organisational learning and process adaptation of agile software development teams resulting from this research should be further applied, evaluated and built in different industrial contexts of software development.

The topic of organisational SPI in the context of agile software development teams includes many interesting issues to be further studied. For example, many organisations still contemplate adopting agile software development methodologies while maintaining their accomplished level of maturity. Thus, the future research should address the relation between the dominating process standards and agile software development. It can be expected that the ongoing standardisation work of agile methodologies will raise unchartered perspectives on this topic.

In addition, various different explanatory studies of testing theories – even from an interdisciplinary point of view (e.g., pedagogical, educational or psychological) – would be needed to provide an understanding of the learning process that takes place within the agile software development teams conducting iterative reflections, and within agile software development organisations conducting bottom-up learning.

References

Abrahamsson, P. 2001. Rethinking the Concept of Commitment in Software Process Improvement. Scandinavian Journal of Information Systems, Vol. 13, pp. 69–98.

Abrahamsson, P. 2002. The Role of Commitment in Software Process Improvement. Doctoral Thesis, University of Oulu.

Abrahamsson, P. 2003. Extreme Programming: First Results from a Controlled Case Study. In: The proceedings of the 29th Euromicro Conference. Belek-Antalya, Turkey. Chroust, G., Hofer, C. & Crnkovic, I. IEEE Computer Society. Pp. 259–266.

Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P. & Salo, O. 2004. Mobile-D: An Agile Approach for Mobile Application Development. In: The proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04). October 24–28, 2004. Vancouver, British Columbia, Canada. Pp. 174–175.

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. 2002. Agile Software Development Methods: Review and Analysis. VTT Publications 478. VTT Electronics. Espoo. 107 p. ISBN 951-38-6009-4; 951-38-6010-8. http://virtual.vtt.fi/inf/pdf/publications/2002/P478.pdf.

Abrahamsson, P., Warsta, J., Siponen, M. T. & Ronkainen, J. 2003. New Directions on Agile Methods: A Comparative Analysis. In: The proceedings of the 25th International Conference on Software Engineering (ICSE'03). IEEE. Pp. 244–254.

Agile Alliance Manifesto for Agile Software Development. 2001. http://www.agilemanifesto.org/principles.html (June, 2006).

Ahern, D. M., Armstrong, J., Clouse, A., Ferguson, J., Hayes, W. & Nidiffer, K. 2005. CMMI® SCAMPI Distilled: Appraisals for Process Improvement. First Edition ed. Addison Wesley Professional. 240 p.

Anacleto, A., von Wangenheim, C. G., Salviano, C. F. & Savi, R. 2004. A Method for Process Assessment in Small Software Companies. In: The proceedings of the 4th International SPICE Conference on Process Assessment and Improvement. April, 2004. Lisbon, Portugal. Pp. 69–76.

Andersen, C. V., Arent, J., Bang, S. & Iversen, J. 2002. Project Assessments. In: *Improving Software Organizations: from Principles to Practice*. Addison-Wesley, Boston. Pp. 167–184.

Arthur, L. J. 1993. Improving Software Quality: An Insider's Guide to TQM. John Wiley & Sons, Inc. New York, 287 p.

Avison, D., Lau, F., Myers, M. & Nielsen, P. A. 1999. Action Research. Communications of the ACM, Vol. 42, 1 (1), pp. 94–97.

Back, R.-J., Milovanov, L. & Porres, I. 2005. Software Development and Experimentation in an Academic Environment: The Gaudi Experience. In: The proceedings of the Product Focused Software Process Improvement (PROFES 2005). June 2005. Oulu, Finland. Springer. Pp. 414–428.

Balzer, R., Cheatham, T. E. & Green, C. 1983. Software Technology in the 1990's: Using a New Paradigm. Computer, Vol. 16 (11), pp. 39–45.

Basili, V. R. 1985. Quantitative Evaluation of Software Methodology. In: The proceedings of the First Pan Pacific Computer Conference. Melbourne, Australia. Pp. 379–398.

Basili, V. R. 1989. Software Development: A Paradigm for the Future. In: The proceedings of the COMPSAC'89. Orlando, Florida. IEEE. Pp. 471–485.

Basili, V. R. 1994. The Goal Question Metric Approach. In: *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc. Pp. 528–532.

Basili, V. R. & Caldiera, G. 1994. Experience Factory. In: *Encyclopedia of Software Engineering*. Marciniak, J. J. (ed.). John Wiley & Sons, Inc. Pp. 469–476.

Basili, V. R. & Caldiera, G. 1995. Improve Software Quality by Reusing Knowledge and Experience. Sloan Management Review, Vol. 37, 1 (1), pp. 55–64.

Basili, V. R. & Reiter, R. 1981. A Controlled Experiment Quantitatively Comparing Software Development Approaches. IEEE Transactions on Software Engineering, Vol. 7, 3 (3), pp. 299–320.

Basili, V. R. & Rombach, H. D. 1987. Tailoring the Software Process to Project Goals and Environments. In: The proceedings of the 9th International Conference on Software Engineering. March 1987. Monterey, CA. Pp. 345–357.

Basili, V. R. & Weiss, D. 1984. A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering, Vol. SE-10, 6 (6), pp. 728–738.

Baskerville, R., Mathiassen, L., Pries-Heje, J. & DeGross, J. I. 2005. Business Agility and Information Technology Diffusion. Springer. 380 p.

Baskerville, R. & Myers, M. D. 2004. Special Issue on Action Research in Information Systems: Making IS Research Relevant to Practice – Foreword. MIS Quarterly, Vol. 28, 3 (3), September 2004, pp. 329–335.

Baskerville, R. L. 1999. Investigating Information Systems with Action Research. Communication of the Association for Information Systems, Vol. 2 (19), October 1999.

Beck, K. 1999. Embracing Change with Extreme Programming. IEEE Computer, Vol. 32, 10 (10), pp. 70–77.

Beck, K. 2000. Extreme Programming Explained: Embrace Change. Addison Wesley Longman, Inc. 190 p.

Beck, K. 2003. Test-Driven Development By Example. 1st ed. Addison-Wesley. 220 p.

Beck, K. & Andres, C. 2004. Extreme Programming Explained: Embrace Change. Second Edition. Addison-Wesley. Boston, 189 p.

Bell Canada *Trillium: Model for Telecom Product Development & Support Process Capability*. Release 3.0. Bell Canada. December, 1994. 118 p.

Benington, H. D. 1983. Production of Large Computer Programs. Annals of the History of Computing, Vol. 5, 4 (4), October, pp. 350–361.

Birk, A., Dingsøyr, T. & Stålhane, T. 2002. Postmortem: Never Leave a Project without It. IEEE Software, Vol. 19, 3 (3), pp. 43–45.

Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. Computer, Vol. 21, 5 (5), May 1988, pp. 61–72.

Boehm, B. & Turner, R., 2003a. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley. 266 p.

Boehm, B. & Turner, R., 2003b. Using Risk to Balance Agile and Plan-Driven Methods. Computer, Vol. 36, 6 (6), June, pp. 57–66.

Boehm, B. & Turner, R. 2005. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. IEEE Software, Vol. 22(5), September–October, pp. 30–39.

Briand, L., El Emam, K. & Melo, W. L. 1999. An Inductive Method for Software Process Improvement: Concrete Steps and Guidelines. In: *Elements of Software Process Assessment & Improvement*. El Emam, K. & Madhavji, N. H. (ed.). IEEE Computer Society: Los Alamitos, California. Pp. 113–130.

Chein, I., Cook, S. W. & Harding, J. 1948. The Field of Action Research. American Psychologist, Vol. 3, pp. 43–50.

Chua, W. F. 1986. Radical Development in Accounting Thought. The Accounting Review, Vol. 61, pp. 601–632.

Coad, P., LeFebvre, E. & De Luca, J. 1999. Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall. 221 p.

Cockburn, A. 1998. Surviving Object-Oriented Projects. Addison-Wesley. Reading, Mass. 250 p.

Cockburn, A. 2002. Agile Software Development. Addison-Wesley. Boston, 278 p.

Cockburn, A. 2005. Crystal Clear: a Human-Powered Methodology for Small Teams. Addison-Wesley. 312 p.

Cohn, M. & Ford, D. 2003. Introducing an Agile Process to an Organization. Computer, Vol. 36 (6), June, pp. 74–78.

Collier, B., DeMarco, T. & Fearey, P. 1996. A Defined Process for Project Post Mortem Review. IEEE Software, Vol. 13, 4 (4), pp. 65–72.

Cooper, H. M. 1984. The Integrative Research Revies: A Systematic Approach. SAGE Publications, Inc. Beverly Hills. 144 p.

Coplien, J. O. & Harrison, N. B. 2005. Organizational Patterns of Agile Software Development. ed. Pearson Prenctice Hall. Upper Saddle River, NJ. 401 p.

Cunningham, J. B. 1997. Case Study Principles for Different Types of Cases. Quality and Quantity, Vol. 31, pp. 401–423.

Davis, L. E. & Taylor, J. T. 1972. Design of Jobs. Penguin. Baltimore. 250 p.

Davison, R. M., Martinsons, M. G. & Kock, N. 2004. Principles of Canonical Action Research. Information Systems Journal, Vol. 14, pp. 65–86.

Debou, C. 1999. Goal-Based Software Process Improvement Planning. In: *Better software practice for business benefit: Principles and experience*. Messnarz, R. & Tully, C. (ed.) IEEE Computer Society: Los Alamitos, CA. Pp. 107–150.

DeMarco, T. 1982. Controlling Software Projects: Management, Measurement and Estimation. Yourdon Press. New York. 296 p.

Deming, W. E. 1990. Out of the Crisis. 10 Printing ed. Massachussetts Institute of Technology, Center for Advanced Engineering Study. Cambridge. 507 p.

Dingsøyr, T. & Hanssen, G. K. 2002. Extending Agile Methods: Postmortem Reviews as Extended Feedback. In: The proceedings of the 4th International Workshop on Learning Software Organizations (LSO'02)). 14. November, 2002. Chicago, Illinois, USA. Pp. 4–12.

DSDMConsortium, 2003. DSDM: Business Focused Development. Addison-Wesley. 239 p.

Dybå, T. 2000. An Instrument for Measuring the Key Factors of Success in Software Process Improvement. Empirical Software Engineering, Vol. 5, pp. 357–390.

Dybå, T., Dingsøyr, T. & Moe, N. B. 2004. Process Improvement in Practice: a Handbook for IT Companies. Kluwer Academic Publishers. Boston. 114 p.

Eckstein, J. 2004. Agile Software Development in Large – Diving into the Deep. Dorset House Publishing. New York, USA. 248 p.

Feigenbaum, A. V. 1991. Total Quality Control. 3rd Edition, Revised (40th Anniversary Edition). McGraw-Hill, Inc. New York, 863 p.

Fenton, N. & Pfleeger, S. L. 1997. Software Metrics: A Rigorous & Practical Approach. Second Edition ed. PWS Publishing Company. Boston. 638 p.

Fitzgerald, B. 1997. The Use of Systems Development Methodologies in Practice: A Field Study. The Information Systems, Vol. 7, 3 (3), pp. 201–212.

Fitzgerald, B. 1998. An Empirical Investigation into the Adoption of Systems Development Methodologies. Information & Management, Vol. 34, 6 (6), pp. 317–328.

Fitzgerald, B., Russo, N. L. & O'Kane, T. 2004. Software Development Method Tailoring at Motorola. Communications of the ACM, Vol. 46, 4 (4), April, pp. 64–70.

Florac, W. A. & Carleton, A. D. 1999. Measuring the Software Process: Statistical Process Control for Software Process Improvement. Addison-Wesley Longman. Massachussetts. 270 p.

Florac, W. A., Carleton, A. D. & Barnard, J. R. 2000. Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process. IEEE Software, Vol. 17, 4 (4). July–August, pp. 97–106.

Garvin, D. A. 1993. Building a Learning Organization. Harvard Business Review, Vol. July–August, pp. 78–89.

Garvin, D. A. 2000. Learning in Action. Harvard Business School Press. Boston, Massachusetts. 256 p.

Gilb, T. 1981. Evolutionary Development. ACM SIGSOFT Software Engineering Notes, Vol. 6, 2 (2). April, pp. 17.

Gilb, T. 1988. Principles of Software Engineering Management. Addison-Wesley. Wokingham, UK, 464 p.

Gilb, T. 2005. Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage. Butterworth-Heinemann. 480 p.

Ginsberg, M. P. & Quinn, L. H. 1995. *Process Tailoring and the Software Capability Maturity Model*. CMU/SEI-94-TR-024. Software Engineering Institute. November. 53 p.

Goldensen, D. R. & Herbsleb, J. D. 1995. *After the Appraisal: A Systematic Survey of Process Improvement, Its Benefits, and Factors that Influence Success.* CMU/SEI-95-TR-009. Software Engineering Institute. Pittsburgh. 50 p.

Gould, P. 1997. What is Agility? Manufacturing Engineer, Vol. February, pp. 28–31.

Grady, R. B. 1992. Practical Software Metrics for Project Management and Process Improvement. Prentice-Hall, Inc. New Jersey. 270 p.

Gremba, J. & Myers, C. 1997. The IDEALSM Model: A Practical Guide for Improvement. Bridge (SEI Publication), Vol. 3 (3).

Hanhineva, A. 2004. Test Driven Development in Mobile Java Environment. Master's Thesis, University of Oulu.

Highsmith, J. 2004. Agile Project Management. Addison-Wesley. 276 p.

Highsmith, J. & Cockburn, A., 2001. Agile Software Development: The Business of Innovation. Computer, Vol. 34, 9 (9), September, pp. 120–122.

Highsmith, J. A. 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing. New York, NY. 358 p.

Hulkko, H. 2004. Pair Programming and its Impact on Software Quality. Master's Thesis, University of Oulu.

Hulkko, H. & Abrahamsson, P. 2005. A Multiple Case Study on the Impact of Pair Programming on Product Quality. In: The proceedings of the 27th International Conference on Software Engineering (ICSE 2005). 15–21 May, 2005. St. Louis, Missouri, USA. Pp. 495–504.

Hult, M. & Lennung, S.-Å. 1980. Towards A Definition of Action Research: A Note and Bibliography. Journal of Management Studies, Vol. 17, pp. 241–250.

Humphrey, W. S. 1995. A Discipline for Software Engineering. Addison Wesley Longman, Inc. 242 p.

Humphrey, W. S. 2000. Introduction to the Team Software Process. Addison Wesley Longman, Inc. Massachussetts, USA. 463 p.

Ihme, T. & Abrahamsson, P. 2005. Agile Architecting: the Use of Architectural Patterns in Mobile Java Applications. International Journal of Agile Manufacturing, Vol. 8, 2 (2), pp. 97–112.

Ishikawa, K. 1985. What is Total Quality Control: the Japanese Way. Prentice-Hall, Inc. 215 p.

ISO/IEC *Information Technology – Software Process Assessment – Part 7: Guide for Use in Process Improvement*. ISO/IEC JTC 1/SC 7. ISO/IEC. 1998-01-12, 1998. 43 p.

Iversen, J., Nielsen, P. A. & Nørbjerg, J. 2002. Problem Diagnosis in SPI. In: *Improving Software Organizations: from Principles to Practice*. Addison-Wesley, Boston. Pp. 153–166.

Jalote, P. 2002. Lessons Learned in Framework-Based Software Process Improvement. In: The proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02). December. Pp. 261–265.

James, T. 2005. Stepping Back from Lean. IEE Manufacturing Engineer, Vol. February/March, pp. 16–21.

Jeffries, R. E. 1999. eXtreme Testing: Why Aggressive Software Development Calls for Radical Testing Efforts. Software Testing & Quality Engineering, Vol. March/April, pp. 23–26.

Järvinen, P. 2001. On Research Methods. Juvenes-Print. Tampere. 189 p.

KBSt – Federal Government Co-Ordination and Advisory Agency *The New V-Modell XT: Development Standard for IT Systems of the Federal Republic of Germany*. 2004.

Keenan, F. 2004. Agile Process Tailoring and probLem analYsis (APTLY). In: The proceedings of the 26th International Conference on Software Engineering (ICSE 2004). 23–28 May, 2004. Edinburgh, Scotland. IEEE. Pp. 45–47.

Kerth, N. L. 2001. Project Retrospectives: A Handbook for Team Reviews. Dorset House Publishing. 268 p.

Kiely, G. & Fizgerald, B. 2005. An Investigation of the Use of Methods within Information Systems Development Projects. The Electronic Journal of Information Systems in Developing Countries (EJISDC), Vol. 22, pp. 1–13.

Klein, H. K. & Myers, M. D. 1999. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. MIS Quarterly, Vol. 23 (1), March, pp. 67–94.

Koch, A. S. 2005. Agile Software Development: Evaluating the Methods for Your Organization. Artech House. Boston. 274 p.

Komi-Sirviö, S. 2004. Development and Evaluation of Software Process Improvement Methods. Doctoral Thesis, University of Oulu. VTT, Espoo. VTT Publications 535. 175 p. + app. 78 p. ISBN 951-38-6388-3; 951-38-6389-1. http://virtual.vtt.fi/inf/pdf/publications/2004/P535.pdf.

Korkala, M. & Abrahamsson, P. 2004. Extreme Programming: Reassessing the Requirements Management Process for an Offsite Customer. In: The proceedings of the European Software Process Improvement Conference (EuroSPI 2004). November, 2004. Trondheim, Norway. Springer. Pp. 12–22.

Koskela, J. & Abrahamsson, P. 2004. On-Site Customer in an XP Project: Empirical Results from a Case Study. In: The proceedings of the European Software Process Improvement Conference (2004). Trondheim, Norway. Springer-Verlag. Pp. 73–82.

Krasner, H. 1999.The Payoff for Software Process Improvement: What it is and How to Get it. In: *Elements of Software Process Assessment & Improvement*. El Emam, K. & Madhavji, N. H. (ed.). IEEE Computer Society: Los Alamitos, California. Pp. 113–130.

Kruchten, P. 1999. The Rational Unified Process. Addison-Wesley. Reading, Massachussetts. 254 p.

Kruchten, P. 2000. The Rational Unified Process: an Introduction. Addison-Wesley Professional. 320 p.

Kuntzmann-Combelles, A., Comer, P., Holdsworth, J. & Shirlaw, S. 1992. AMI Handbook: a Quantitative Approach to Software Management. GEC-Marconi Research Centre. 170 p.

Kuvaja, P. & Bicego, A. 1993. Bootstrap: Europe's Assessment Method. IEEE Software, Vol. 10, 3 (3), May, pp. 93–95.

Kuvaja, P., Similä, J., Kranik, L., Bicego, A., Saukkonen, S. & Koch, G. 1994. Software Process Assessment and Improvement: The BOOTSTRAP Approach. Blackwell Publishers. Oxford, UK. 149 p.

Kyllönen, P. 2005. A Framework for Managing Agile Projects. Master's Thesis, University of Oulu.

Kähkönen, T. 2005. Life Cycle Model for Software Process Improvement Project Deploying an Agile Method. In: The proceedings of the International Conference on Agility (ICAM 2005). July, 2005. Helsinki, Finland. Pp. 225–232.

Kähkönen, T. & Abrahamsson, P. 2004. Achieving CMMI Level 2 with Enhanced Extreme Programming Approach. In: The proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004). April, 2004. Kansai Science City, Japan. Springer. Pp. 378–392.

Lappo, P. & Andrew, H. C. T. 2004. Assessing Agility. In: The proceedings of the Extreme Programming and Agile Processes in Software Engineering. June, 2004. Garmisch-Partenkirchen, Germany. Eckstein, J. & Baumeister, H. (ed.). Springer. Pp. 331–338.

Larman, C. 2004. Agile and Iterative Development: A Manager's Guide. Pearson Education, Inc. Boston. 342 p.

Larman, C. & Basili, V. R. 2003. Iterative and Incremental Development: A Brief History. IEEE Software, Vol. 20, pp. 47–56.

Lau, F. 1999. Toward a Framework for Action Research in Information Systems Studies. Information, Technology & People, Vol. 12, 2 (2), pp. 148–175.

Lewin, K. 1946. Action Research and Minority Problems. Journal of Social Issues, Vol. 2 (2), pp. 34–46.

Lindvall, M., Basili, V. R., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. & Zelkowitz, M. V. 2002. Empirical Findings in Agile Methods. In: The proceedings of the XP/Agile Universe 2002. August 4–7. Chicago, IL, USA. Williams, D. W. (ed.). L. A. Springer. Pp. 197–207.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J. & Kähkönen, T. 2004. Agile Software Development in Large Organizations. Computer, Vol. 37, 12 (12), December, pp. 26–34.

Lycett, M., Macredie, R. D., Patel, C. & Paul, R. J. 2003. Migrating Agile Methods to Standardized Development Practice. Computer, Vol. 36, 6 (6), June, pp. 79–85.

Malouin, J. L. & Landry, M. 1983. The Miracle of Universal Methods in Systems Design. Journal of Applied Systems Analysis, Vol. 10, pp. 47–62.

March, S. T. & Smith, G. F. 1995. Design and Natural Science Research on Information Technology. Decision Support Systems, Vol. 15, pp. 251–266.

Mathiassen, L., Pries-Heje, J. & Nqwenyama, O. 2002. Improving Software Organizations: from Principles to Practice. Addison-Wesley. Boston. 336 p.

McCracken, D. D. & Jackson, M. A. 1982. Life Cycle Concept Considered Harmful. Software Engineering Notes, Vol. 7, 2 (2), April, pp. 29–32.

McFeeley, B. *IDEAL(SM): A Users Guide for Software Process Improvement*. CMU/SEI-96-HB-001. Software Engineering Institute (SEI). February 1996. 222 p.

Merriam-Webster Online Dictionary. http://www.m-w.com/ (June 2006).

Miles, M. B. & Huberman, A. M. 1994. Qualitative Data Analysis: An Expanded Sourcebook. Second Edition. SAGE Publications, Inc. Thousand Oaks. 338 p.

Morgan, D. L. 1984. Focus Groups: A New Tool for Qualitative Research. Qualitative Sociology, Vol. 7(3), Fall 1984, pp. 253–270.

Morgan, D. L. 1988. Focus Groups as Qualitative Research. SAGE Publications. Newbury Park. 85 p.

Morien, R. 2005. Agile Management and the Toyota Way for Software Project Management. In: The proceedings of the 3rd International Conference on Industrial Informatics (INDIN). August 10, 2005. Perth, WA, Australia. IEEE. Pp. 516–522.

NATO Science Committee. 1969. *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*. Naur, P. & Randell, B. (eds.). Scientific Affairs Division, NATO. Garmisch, 7–11 Oct., 1968. 231 p.

Nawrocki, J., Walter, B. & Wojciechowski, A. 2001. Toward Maturity Model For Extreme Programming. In: The proceedings of the 27th Euromicro Conference 2001. 4–6 September 2001. Warsaw, Poland. Pp. 233–239.

Niazi, M., Wilson, D. & Zowghi, D. 2003. A Model for the Implementation of Software Process Improvement: A Pilot Study. In: The proceedings of the Third International Conference on Quality Software (QSIC'03). November 2003. Pp. 196–203.

Nielsen, P. A. & Pries-Heje, J. 2002. A Framework for Selecting an Assessment Strategy. In: *Improving Software Organizations: from Principles to Practice*. Addison-Wesley, Boston. Pp. 185–197.

Nonaka, I. & Hirotaka, T. 1995. The Knowledge-Creating Company. Oxford University Press, Inc. 284 p.

Oivo, M. & Basili, V. R. 1992. Representing Software Engineering Models: The TAME Goal Oriented Approach. IEEE Transactions on Software Engineering, Vol. 18 (10), October, pp. 886–898.

Oleson, J. D. 1998. Pathways to Agility: Mass Customization in Action. John Wiley & Sons, Inc. New York, USA. 262 p.

Patton, M. Q. 2002. Quatlitative Research & Evaluation Methods. Sage Publications, Inc. Thousand Oaks, California. 598 p.

Paulk, M., Curtis, B., Chrissis, M. & Weber, C. 1993. *Capability Maturity Model for Software (Version 1.1)*. CMU/SEI-93-TR-024. Software Engineering Institute (SEI). February. 65 p.

Paulk, M. C. 2001. Extreme Programming from a CMM Perspective. Software, Vol. 18, 6 (6), Nov.–Dec, pp. 19–26.

Pfleeger, S. L. & Rombach, H. D. 1994. Measurement Based Process Improvement. IEEE Software, Vol. 11, 4 (4), July, pp. 8–11.

Pikkarainen, M. & Passoja, U. 2005. An Approach for Assessing Suitability of Agile Solutions: A Case Study. In: The proceedings of the Sixth International Conference on Extreme Programming and Agile Processes in Software Engineering. Sheffield University, UK. Pp. 171–179.

Poppendieck, M. & Poppendieck, T. 2003. Lean Software Development: An Agile Toolkit. Addison-Wesley. Boston. 203 p.

Rapoport, R. N. 1970. Three dilemmas of action research. Human Relations, Vol. 23, pp. 499–513.

Rawlinson, J. G. 1981. Creative Thinking and Brainstorming. Gower Publishing Company Limited. Westmead. 128 p.

Reifer, D. J. 2003. XP and the CMM. IEEE Software, Vol. 20, 3 (3), May/June, pp. 14–15.

Rico, D. F. 2004. ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers. J. Ross Publishing. Florida, U.S.A. 218 p.

Ross, A. & Francis, D. 2003. Lean is Not Enough. IEE Manufacturing Engineer, Vol. August/September, pp. 14–17.

Rout, T. P., Tuffley, A., Cahill, B. & Hodgen, B. 2000. The Rapid Assessment of Software Process Capability. In: The proceedings of the First International Conference on Software Process Improvement and Capability dEtermination (SPICE 2000). 10th–11th June, 2000. Limerick, Ireland. Pp. 47–55.

Royce, W. W. 1970. Managing the Development of Large Software Systems. In: The proceedings of the WESCON. San Francisco. IEEE CS. Pp. 328–339.

Schwaber, C. & Fichera, R. 2005. *Corporate IT Leads The Second Wave of Agile Adoption*. Forrester Research, Inc. November 30, 2005. 6 p.

Schwaber, K. 1995. Scrum Development Process. In: The proceedings of the OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag. Pp. 117–134.

Schwaber, K. 2004. Agile Project Management with Scrum. Microsoft Press. Washington. 163 p.

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. Prentice-Hall. Upper Saddle River, NJ. 158 p.

Scupin, R. 1997. The KJ Method: A Technique for Analyzing Data Derived from Japanese Ethnology. Human Organization, Vol. 56, 2 (2), pp. 233–237.

SEI, C. M. S. E. I. *Capability Maturity Model® Integration (CMMI^{SM}), Version 1.1*. Carnegie Mellon Software Engineering Institute. 2001.

Sillitti, A., Janes, A., Succi, G. & Vernazza, T. 2003. Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. In: The proceedings of the 29th EUROMICRO Conference. Pp. 336–342.

Stapleton, J. 2003. DSDM: Business Focused Development. Second Edition. Addison Wesley. London. 239 p.

Stålhane, T., Dingsøyr, T., Hanssen, G. K. & Moe, N. B. 2001. Post Mortem – An Assessment of Two Approaches. In: The proceedings of the European Software Process Improvement. Limerick, Ireland. Pp. 129–141.

Susman, G. I. & Evered, R. D. 1978. An Assessment of the Scientific Merits of Action Research. Administrative Science Quarterly, Vol. 23, pp. 582–603.

Sweeney, A. & Bustard, D. W. 1997. Software Process Improvement: Making it Happen in Practice. Software Quality Journal, Vol. 6 (4), pp. 265–274.

van Solingen, R. & Berghout, E. 1999. The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. The McGraw-Hill Companies. 199 p.

Wilkie, F. G., McFall, D. & McCaffery, F. 2005. An Evaluation of CMMI Process Areas for Small- to Medium-sized Software Development Organizations. Software Process Improvement and Practice, Vol. 10, 2 (2), pp. 189–201.

Williams, L. & Cockburn, A. 2003. Agile Software Development: It's about Feedback and Change. IEEE Computer Society, Vol. 36, 6 (6), June, pp. 39–43.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. 2000. Experimentation in Software Engineering. Kluwer Academic Publishers. Boston. 204 p.

Vriens, C. 2003. Certifying for CMM Level 2 and ISO9001 with XP@Scrum. In: The proceedings of the Agile Development Conference (ADC'03). September, 2003. IEEE Computer Society. Pp. 120–124.

Yin, R. K. 2003. Case Study Research: Design and Methods. Third Edition. SAGE Publications. Thousand Oaks, 5. 181 p.

Zahran, S. 1998. Software Process Improvement: Practical Guidelines for Business Success. Addison-Wesley. 447 p.

Published by

Series title, number and report code of publication

VTT Publications 618
VTT–PUBS–618

Author(s)
Salo, Outi

Title

Enabling Software Process Improvement in Agile Software Development Teams and Organisations

Abstract

Agile software development has challenged the traditional ways of delivering software as it provides a very different approach to software development. In recent decades, software process improvement (SPI) has been widely studied in the context of traditional software development, and its strengths and weaknesses have been recognised. As organisations increasingly adopt agile software development methodologies to be used alongside traditional methodologies, new challenges and opportunities for SPI are also emerging. One challenge is that traditional SPI methods often emphasise the continuous improvement of organisational software development processes, whereas the principles of agile software development focus on iterative adaptation and improvement of the activities of individual software development teams to increase effectiveness.

The focus of this thesis is twofold. The first goal is to study how agile software development teams can conduct SPI, according to the values, principles and practices of agile software development, in tandem with the success factors of traditional SPI. The second goal is to study how the team-centred SPI of agile software development and the traditional view of organisational improvement can be integrated to co-exist in a mutually-beneficial manner in software development organisations. The main research methodology in this thesis is action research (AR). The empirical data is taken from six agile software development case projects. The results of this research have been published in a total of seven conference, and journal, papers.

Keywords
software process improvement, SPI, agile software development, iterative improvement process

ISBN
951–38–6869–9 (soft back ed.)
951–38–6870–2 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN	Project number
VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/publications/index.jsp)	

Date	Language	Pages	Price
November 2006	English	149 p. + app. 96 p.	F

Name of project	Commissioned by

Contact	Sold by
VTT Technical Research Centre of Finland P.O. Box 1100, FI-90571 OULU, Finland Phone internat. +358 20 722 111 Fax +358 20 722 2320	VTT Technical Research Centre of Finland P.O.Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4404 Fax +358 20 722 4374

Agile software development has challenged the traditional ways of delivering software as it provides a very different approach to software development. In recent decades, software process improvement (SPI) has been widely studied in the context of traditional software development, and its strengths and weaknesses have been recognised. As organisations increasingly adopt agile software development methodologies to be used alongside traditional methodologies, new challenges and opportunities for SPI are also emerging. One challenge is that traditional SPI methods often emphasise the continuous improvement of organisational software development processes, whereas the principles of agile software development focus on iterative adaptation and improvement of the activities of individual software development teams to increase effectiveness.

The focus of this thesis is twofold. The first goal is to study how agile software development teams can conduct SPI, according to the values, principles and practices of agile software development, in tandem with the successful factors of traditional SPI. The second goal is to study how the team-centred SPI of agile software development and the traditional view of organisational improvement can be integrated to co-exist in a mutually-beneficial manner in software development organisations. The main research methodology in this thesis is action research (AR). The empirical data is taken from six agile software development case projects. The results of this research have been published in a total of seven conference, and journal, papers.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT PL 1000 02044 VTT Puh. 020 722 4404 Faksi 020 722 4374	VTT PB 1000 02044 VTT Tel. 020 722 4404 Fax 020 722 4374	VTT P.O. Box 1000 FI-02044 VTT, Finland Phone internat. +358 20 722 4404 Fax +358 20 722 4374

ISBN 951-38-6869-9 (soft back ed.)
ISSN 1235-0621 (soft back ed.)
ISBN 951-38-6870-2 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455-0849 (URL: http://www.vtt.fi/inf/pdf/)