IMPLEMENTING LEAN AND AGILE SOFTWARE DEVELOPMENT IN INDUSTRY

Kai Petersen

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2010:04

School of Computing

Implementing Lean and Agile Software Development in Industry

Kai Petersen

Blekinge Institute of Technology Doctoral Dissertation Series
No 2010:04

Implementing Lean and Agile Software Development in Industry

Kai Petersen

School of Computing
Blekinge Institute of Technology
SWEDEN

School of Computing
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2010
ISBN 978-91-7295-180-8
Blekinge Institute of Technology Doctoral Dissertation Series
ISSN 1653-2090
urn:nbn:se:bth-00465

Q: What are the most exciting, promising software engineering ideas or techniques on the horizon?

A: I don't think that the most promising ideas are on the horizon. They are already here and have been for years, but are not being used properly.

–David L. Parnas

Abstract

Background: The software market is becoming more dynamic which can be seen in frequently changing customer needs. Hence, software companies need to be able to quickly respond to these changes. For software companies this means that they have to become agile with the objective of developing features with very short lead-time and of high quality. A consequence of this challenge is the appearance of agile and lean software development. Practices and principles of agile software development aim at increasing flexibility with regard to changing requirements. Lean software development aims at systematically identifying waste to focus all resources on value adding activities.

Objective: The objective of the thesis is to evaluate the usefulness of agile practices in a large-scale industrial setting. In particular, with regard to agile the goal is to understand the effect of migrating from a plan-driven to an agile development approach. A positive effect would underline the usefulness of agile practices. With regard to lean software development the goal is to propose novel solutions inspired by lean manufacturing and product development, and to evaluate their usefulness in further improving agile development.

Method: The primary research method used throughout the thesis is case study. As secondary methods for data collection a variety of approaches have been used, such as semi-structured interviews, workshops, study of process documentation, and use of quantitative data.

Results: The agile situation was investigated through a series of case studies. The baseline situation (plan-driven development) was evaluated and the effect of the introduction of agile practices was captured, followed by an in-depth analysis of the new situation. Finally, a novel approach, Software Process Improvement through the Lean Measurement (SPI-LEAM) method, was introduced providing a comprehensive measurement approach supporting the company to manage their work in process and capacity. SPI-LEAM focuses on the overall process integrating different dimensions (requirements, maintenance, testing, etc.). When undesired behavior is observed a drill-down analysis for the individual dimensions should be possible. Therefore, we provided solutions for the main product development flow and for software maintenance. The lean solutions were evaluated through case studies.

Conclusion: With regard to agile we found that the migration from plan-driven to agile development is beneficial. Due to the scaling of agile new challenges arise with the introduction. The lean practices introduced in this thesis were perceived as useful by the practitioners. The practitioners identified concrete decisions in which the lean solutions could support them. In addition, the lean solutions allowed identifying concrete improvement proposals to achieve a lean software process.

Acknowledgements

First and foremost, I would like to thank my supervisor and collaborator Professor *Claes Wohlin* for his support and feedback on my work, for the fruitful collaboration on papers, and for always responding to my questions at any time of day or night.

I would also like to thank all colleagues at *Ericsson AB* for participating in my studies. These include participants in interviews, workshops, and in the TiQ analysis team. All of you have provided valuable input and feedback to my work despite your busy schedules, for which I am thankful. In particular, I also would like to express my gratitude to *Eva Nilsson* at Ericsson for her strong support and thus for making the implementations of the solutions presented in this thesis possible, and for providing important feedback. Thanks also go to *PerOlof Bengtsson* for his continuous support and commitment throughout my studies.

The support from my friends and colleagues from the SERL and DISL research groups is also highly appreciated. In particular, I would like to thank *Dejan Baca* and *Shahid Mujtaba* for their collaboration on papers, and for the good times after work.

Last but not least, I would like to express my sincere gratitude to my mother *Frauke* for always supporting me in what I wanted to achieve against all obstacles, and to my brother *Stephan* for always being there.

This work was funded jointly by Ericsson AB and the Knowledge Foundation in Sweden under a research grant for the project “Blekinge - Engineering Software Qualities (BESQ)” (http://www.bth.se/besq).

Overview of Papers

Papers included in this thesis.

Chapter 2. Kai Petersen.
'Is Lean Agile and Agile Lean? A Comparison Between Two Development Paradigms', To be published in: Modern Software Engineering Concepts and Practices: Advanced Approaches; Ali Dogru and Veli Bicer (Eds.), IGI Global, 2010

Chapter 3. Kai Petersen, Claes Wohlin, and Dejan Baca.
'The Waterfall Model in Large-Scale Development', *Proceedings of the 10th International Conference on Product Focused Software Development and Process Improvement (PROFES 2009)*, Springer, Oulu, Finland, pp. 386-400, 2009.

Chapter 4. Kai Petersen and Claes Wohlin.
'The Effect of Moving from a Plan-Driven to an Incremental and Agile Development Approach: An Industrial Case Study', Submitted to a journal, 2009.

Chapter 5. Kai Petersen and Claes Wohlin.
'A Comparison of Issues and Advantages in Agile and Incremental Development Between State of the Art and Industrial Case'. *Journal of Systems and Software*, 82(9), pp. 1479-1490, 2009.

Chapter 6. Kai Petersen.
'An Empirical Study of Lead-Times in Incremental and Agile Development', *To appear in: Proceedings of the International Conference on Software Process (ICSP 2010)*, 2010.

Chapter 7. Kai Petersen and Claes Wohlin.
'Software Process Improvement through the Lean Measurement (SPI-LEAM) Method'. *Journal of Systems and Software*, in print, 2010.

Chapter 8. Kai Petersen and Claes Wohlin.
'Measuring the Flow of Lean Software Development', *Software: Practice and Experience*, in print, 2010.

Chapter 9. Kai Petersen.
'Lean Software Maintenance', submitted to a conference, 2010.

Papers that are related to but not included in this thesis.

Paper 1. Kai Petersen.
'A Systematic Review of Software Productivity Measurement and Prediction', submitted to a journal, 2010.

Paper 2. Dejan Baca and Kai Petersen.
'Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec)', To appear in: Proceedings of the International Conference on Product Focused Software Development and Process Improvement (PROFES 2010), 2010.

Paper 3. Shahid Mujtaba, Kai Petersen, and Robert Feldt.
'A Comparative Study Between Two Industrial Approaches for Realizing Software Product Customizations', submitted to a conference, 2010.

Paper 4. Dejan Baca, Bengt Carlsson, and Kai Petersen.
'Static analysis as a security touch point: an industrial case study', submitted to a journal, 2010.

Paper 5. Shahid Mujtaba, Robert Feldt, and Kai Petersen.
'Waste and Lead Time Reduction in a Software Product Customization Process with Value Stream Maps', *To appear in: Proceedings of the Australian Software Engineering Conference (ASWEC 2010)*, IEEE, Auckland, New Zealand, 2010.

Paper 6. Kai Petersen and Claes Wohlin.
'Context in Industrial Software Engineering Research', *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, IEEE, Florida, USA, pp. 401-404, 2009.

Paper 7. Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg.
'Static Code Analysis to Detect Software Security Vulnerabilities: Does Experience Matter?', *Proceedings of the 3rd International Workshop on Secure Software Engineering (SecSE 2009)*, IEEE, Fukuoka, Japan, pp. 804-810, 2009.

Paper 8. Joachim Bayer, Michael Eisenbarth, Theresa Lehner, and Kai Petersen.
'Service Engineering Methodology', *In: Semantic Service Provisioning, Eds. Dominik Kuropka, Peter Tröger, Steffen Staab* , Springer, April 2008.

Paper 9. Kai Petersen and Claes Wohlin.
'Issues and Advantages of Using Agile and Incremental Practices: Industrial Case Study vs. State of the Art', *Proceedings of the 8th Software Engineering Research and Practice Conference in Sweden (SERPS 2008)*, Karlskrona, Sweden, 2008. Chapter 5 is an extension of this paper.

Paper 10. Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson.
'Systematic Mapping Studies in Software Engineering', *Proceedings of the 12th International Conference on Empirical Assessment and Evaluation in Software Engineering (EASE 2008)*, British Computer Society, Bari, Italy, pp. 71-80, 2008.

Paper 11. Kai Petersen, Kari Rönkkö, and Claes Wohlin.
'The Impact of Time Controlled Reading on Software Inspection Effectiveness and Efficiency: A Controlled Experiment', *Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement (ESEM 2008)*, ACM, Kaiserslautern, Germany, pp. 139-148, 2008.

Paper 12. Kai Petersen, Johannes Maria Zaha, and Andreas Metzger.
'Variability-Driven Selection of Services for Service Compositions', *Proceedings of the ICSOC Workshops 2007*, Springer, Vienna, Austria, pp. 388-400, 2007.

Table of Contents

Chapter 1

Introduction

1.1 Preamble

The market for software is fast paced with frequently changing customer needs. In order to stay competitive companies have to be able to react to the changing needs in a rapid manner. Failing to do so often results in a higher risk of market lock-out [34], reduced probability of market dominance [15], and it is less likely that the product conforms to the needs of the market. In consequence software companies need to take action in order to be responsive whenever there is a shift in the customers' needs on the market. That is, they need to meet the current requirements of the market, the requirements being function or quality related. Two development paradigms emerged in the last decade to address this challenge, namely agile and lean software development.

Throughout this thesis agile software development is defined as a development paradigm using a set of principles and practices allowing to respond flexibly and quickly to changes in customers' needs. All agile methods, such as the most prominent ones (SCRUM and eXtreme programming), are driven by similar values. The individual methods differ with regard to the selection of their practices. The values driving the agile community are (1) *"Individuals and interactions over processes and tools"*, (2) *"Working software over comprehensive documentation"*, (3) *"Customer collaboration over contract negotiation"*, and *"Responding to change over following a plan"* [17].

We define the lean software development paradigm as a set of principles and practices focused on the removal of waste leading to a lean software development process. Waste thereby is defined as everything that does not contribute to the value creation for the customer. It is important to acknowledge that it is challenging to define and quan-

tify value, which is a relatively new research field on its own referred to as value-based software engineering [4]. In the field of value-based software engineering the focus is not only on the customer, but concerns all stakeholders involved in the creation of software. Thereby, different stakeholders may value certain aspects of software development (e.g. quality attributes of the software product) differently (cf. [2]). However, in lean manufacturing and software engineering the main focus has been on the value created for the customer (see [23] and Chapter 2). In Chapter 2 we also provide a description of different wastes that are considered as not contributing to the value of the customer, and hence should be removed.

Lean software engineering received much attention primarily from industry after the publication of a book on lean software development by Poppendieck and Poppendieck [23]. The Poppendiecks have practical experience from lean manufacturing and product development. When Mary Poppendick got to k now about the waterfall model in software development, she recognized that the software community could benefit from the ideas of flexible production processes. This was the motivation for them to look into how the lean principles and practices from product development and manufacturing could be used in the case of software development. The main ideas behind lean are to focus all development effort on value adding activities from a customers' perspective and to systematically analyze software processes to identify the waste and then remove it [21, 23], as is reflected in the definition.

The thesis investigates the implementation of lean and agile practices in the software industry. The primary research method used throughout the thesis is case study [38, 29]. The case is a development site of Ericsson AB, located in Sweden. The company produces software applications in the telecommunication and multimedia domain. At the case company agile practices were first introduced. That is, the company studied is coming from a plan-driven approach and step-wise moved towards an agile development approach. In order to further improve the agile approach in place lean analysis tools and measurements were used with the goal of further improving the companies' ability to respond to changes in the market needs.

In relation to the introduction of the lean and agile development approaches the thesis makes two contributions. As the *first contribution* (from hereon referred to as Contribution I) of the thesis, the impact of migrating from a plan-driven to an agile development approach is investigated empirically. For this, first the baseline situation (plan-driven development) is analyzed empirically. Secondly, the effect of the migration from a plan-driven to an agile approach is investigated. Finally, the new situation is analyzed in more depth. As the *second contribution* (from hereon referred to as Contribution II), new solutions inspired by lean ideas are proposed and their usefulness is evaluated in industrial case studies to further improve the processes established in the migration from plan-driven to agile development. A bridge between the two contribu-

tion is built by comparing the goals, principles, practices, and processes of lean and agile software development.

As this thesis focuses on the implementation of lean as well as agile software development it is important to mention that both paradigms share principles and practices, but that there are some distinguishing characteristics as well. When focusing on the lean implementation principles and practices unique to the lean software development paradigm are investigated to determine whether these provide additional benefits to agile software development.

The introduction is synthesizing the individual contributions of the papers included. Section 1.2 provides background information on the investigated software development paradigms. Section 1.3 identifies research gaps in the related work and discusses how the individual studies address these research gaps. The research questions arising from the research gaps are stated and motivated in Section 1.4. The research method used to answer the research questions is presented in Section 1.5. The results (Section 1.6) summarize the outcomes of the individual studies and are the input for synthesizing the evidence from the studies for the lean and agile part 1.7). Finally, Section 1.8 concludes the introduction.

1.2 Background

Three development paradigms are investigated in this thesis, namely plan-driven software development, agile software development, and lean software development. This section provides a brief description of the studied paradigms.

1.2.1 Plan-Driven Software Development

Plan-driven software development is focused on planning everything from the start of a project (as suggested by the name). In addition, the plan-driven approach is characterized as very documentation centric with designated responsibilities for the individual software development disciplines (e.g. requirements engineering, architecture design, implementation, and quality assurance).

The most prominent instantiation of plan-driven software development is the waterfall process as suggested by Royce [28], the process being shown in Figure 1.1. The waterfall process is executed sequentially, following steps representing the different software development disciplines. The goal Royce was pursuing was to provide some structure for executing software processes by assigning the disciplines to distinct software process phases. When a phase is completed the product of that phase is handed over to the following phase, e.g. when the software requirements are specified the

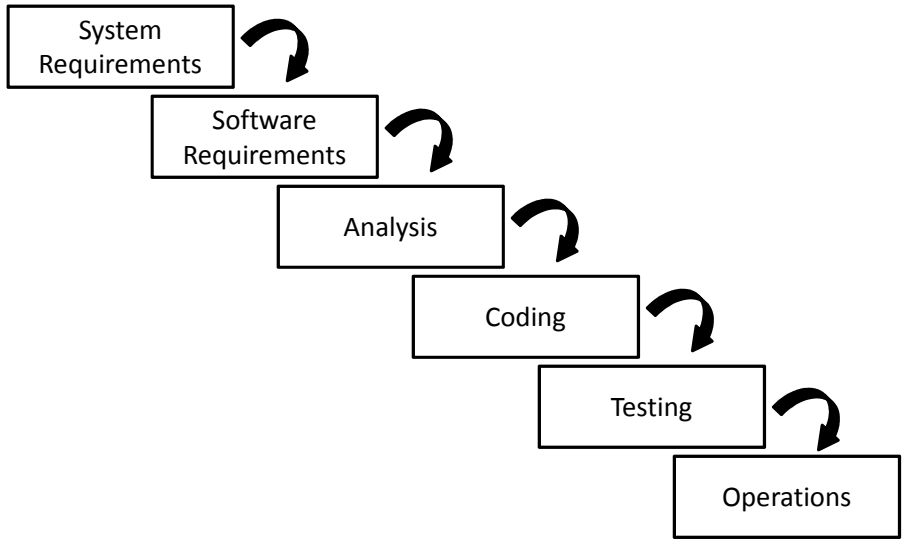


Figure 1.1: Waterfall Process According to Royce

specification is handed over to the analysis phase. It is also apparent that, according to the specification of the waterfall model, each phase only knows the input its preceding phase. For example, the input for testing is the developed code.

The rational unified process (RUP) [16] is another representative of plan-driven approaches. RUP is more relaxed when it comes to the sequence in which the disciplines are executed. That is, the engineering disciplines defined by the process (business modeling, requirements specification, analysis and design, implementation, test, and deployment) are executed throughout the overall development life-cycle consisting of inception, elaboration, construction, and transition. In the inception phase and early elaboration phase the main activities are business modeling and requirements, while these receive less attention in the construction and transition phase. Hence, one can say that the disciplines are overlapping, but have different emphasis depending on the development phase. RUP also proposes several plans to be documented, such as measurement plans, risk management plans, plans for resolving problems, and plans for the current and upcoming iteration. The process is also supported by Rational products

providing tools for modeling and automation aspects related to the software process (e.g. Rational DOORS [14] for requirements management, and ClearCase [35]/ ClearQuest [5] for change and configuration management). To support the tailoring of the RUP process to specific company needs the Rational Method Composer supports in the composition of methods and tools.

The V-Model [30] can be seen as an extension of the waterfall process by mapping verification and validation activities to each sequential development step. That is, the delivered product is verified through operation and supported by maintenance activities, the specification of the requirements is verified through acceptance and system integration testing, and the detailed design and coding activities are verified through unit and component testing.

1.2.2 Agile Software Development

The agile approach stands in strong contrast to the plan-driven methods. This becomes particularly visible considering the four values of agile software development. *"Individuals and interactions over processes and tools"* conflicts with plan-driven as plan-driven prefers strictly separated phases and the communication between phases relies heavily on documentation. In addition the value contradicts RUP as this process relies heavily on planning. *"Working software over comprehensive documentation"* puts an emphasis on the implementation phase and stresses the importance to create a working and valuable product early in the process, which would not be possible in waterfall development where each phase has to be complete before the next one can be started, i.e. deliveries in small increments and iterations are not considered. *"Customer collaboration over contract negation"* contradicts plan-driven development where the requirements specification is a measure of whether the contract has been fulfilled, the requirements specification being established early in the process. In agile development the content of the contract is not set in stone in the beginning as the practice of software engineering has shown that the specification changes continuously. Hence, agile proposes that the customer should be continuously involved throughout the process. *"Responding to change over following the plan"* is not supported by plan-driven development as any change would require a tremendous effort. Looking at the waterfall process in Figure 1.1 a change in the requirements when being in the test phase would lead to changes in the analysis, and coding phase. Working in small increments and iterations allows much shorter feedback cycles and with that supports the ability to change, as proposed by agile development. In Table 1.1 the differences between plan-driven and agile developed are summarized.

Table 1.1: Contrasting Plan-Driven and Agile Development (Inspired by [12])

Aspect	Plan-Driven	Agile
Assumption	Problem is well understood and the desired output is well defined from the beginning.	The desired output is not known completely until the solution is delivered.
Process Models	Waterfall, Rational Unified Process, V-Model	Incremental and iterative, eXtreme programming, SCRUM, Crystal, and other agile models.
Planning	Detailed planning of time-line with clearly defined products and documents to be delivered.	High-level plan for the overall product development life-cycle with details only planned for current iterations.
Requirements engineering	Clearly defined specification phase; requirements specification of overall product with sign-off; detailed requirements specification often part of the contract, requirements change is a formal and work intensive process.	Welcoming change to requirements specs leading to continuous evolution; relaxed change request process; communication with customer over detailed product specifications.
Architecture	Specification of architecture and designs is comprehensive and detailed; Architecture design concentrated on one phase.	Minimal draft of architecture and design specification and re-evaluation of architecture continuously throughput development life-cycle.
Implementation	Programming work concentrated in one phase and coders concentrate mainly on the programming task; programming is specification driven.	Programming work throughout the entire project; programmers have the possibility to interact with customers; collective code ownership and egoless programming; pair programming.
Testing	Testing activities at the end of the implementation phase (big-bang integration); Testers are specialists mainly responsible for testing.	Testing activities throughout development life-cycle (developers have test responsibility); tests also specified and executed by end users.
Reviews and inspections	Formal roles in the review process (e.g. inspection); Use of quality doors to approve software artifacts for hand over between phases.	No explicit roles for reviews and inspections; no formal reviews (except in e.g. feature driven development, but not as formal as the inspection process)

1.2.3 Lean Software Development

The ideas of lean software development are drawn from lean manufacturing and lean product development. Lean manufacturing focused solely on the production process

and focused on removing waste from the production process. Waste is defined as everything that does not contribute to the creation of value for the customer. In production seven types of waste were defined (physical transportation of intermediate products, inventory in form of unfinished work in process, motion of people and equipment, waiting time, over production, over processing by executing activities that are not value adding, and defects) [37]. In lean product development not only the manufacturing process for mass production was in the center of attention, but the overall product development process [21], from creating an idea for the product and specifying it until the delivery of the final product. Hence, lean product development is somewhat closer to software engineering than the pure manufacturing view. Much attention in practice was generated by Mary and Tom Poppendieck who published several books (cf. [23, 24, 25]) on the translation and application of lean manufacturing and product development practices to a software engineering context. Lean software development shares many principles and practices with agile software development. However, at the same time there are differences making the potential for the complementary use of the two paradigms explicit. For example, if a practice found in lean is not included in agile it might be a useful complement. The detailed comparison of the two paradigms is provided in Chapter 2.

1.3 Research Gaps and Contributions

The research gaps have been identified based on the literature reviews of the individual studies. Overall, studying the literature on agile software development we observed that:

- The majority of empirical studies of sufficient quality focused on a single process model, namely eXtreme programming (XP) [3] as identified in the systematic review by Dybå and Dingsøyr [8].

- Studies were focused on projects with development teams, but do not take pre- and post-project activities into account, as they are often found in larger-scale software development. For example, pre-project activities are related to product management and to distributing requirements to development projects. Post-project activities are concerned with integration of individual project results and the release of the software product [11].

- The studies on agile often focused on agile implementations in a smaller scale (focus on teams with a size range from 4 to 23 [8]).

- Mostly immature agile implementations have been studied, which means that we know little about how very experienced organizations perform [8].

- When studying agile the baseline situation is not presented which makes it hard to judge the effect/ improvement potential of agile development in comparison to other models [8].

This makes the general need for studying agile implementations in a large-scale environment under consideration of the overall development life-cycle explicit. The first bullet is partially addressed in this thesis as we do not investigate a particular agile process (such as XP, or SCRUM [31]). Instead, an agile implementation based on a selection of practices that a company found most useful in its specific context (large-scale, market-driven) is investigated. The second bullet is covered in this thesis as we take the end-to-end perspective into account, covering the overall software development life-cycle. The third bullet is addressed by investigating an agile software process where the products developed include more than 500 people overall at the development site of the case company, with multiple projects being run in parallel. The fourth bullet is not addressed as the studied company was in the migration from a plan-driven to an agile process. The fifth bullet is addressed by investigating the baseline situation in detail, then investigating the effect of the migration, and finally taking a closer look at the new situation with the agile practices in place.

With regard to lean it can be observed that lean practices have been well described in a software engineering context in the books by Poppendieck and Poppendieck [23, 24, 25]. However, little empirical evidence on their usefulness in the software engineering context has been provided so far. That is, Middleton [19] studied the lean approach in industry with two teams of different experiences, implementing the principle of building quality in (i.e. assuring quality of the product as early as possible in the development process) by stopping all work and immediately correcting defects, which was the main practice introduced. Another study of Middleton showed lessons learned from introducing several lean practices in the organization (such as minimizing inventory, balance work-load, elimination of rework, and standard procedures). Results were briefly presented in lessons learned. Overall the study of related work showed that there was only one study reporting on the impact of a lean implementation [20]. However, to the best of our knowledge no studies on lean software development have been conducted using the case study approach, and discussing elemental parts of empirical work (such as validity threats, data collection approaches, etc.). Hence, this makes the need explicit to (1) provide and tailor solutions of lean manufacturing/product development to the specific needs in the software industry that are not as explicitly addressed in agile development, and (2) use empirical methods to show their merits and limitations.

As mentioned earlier the thesis makes two contributions, namely the investigation of the effect of moving from plan-driven to agile (Contribution I), and from agile to lean software development (Contribution II). The two contributions are realized through sub-contributions provided by the individual studies. For Contribution I studies S2 to S5 provide the sub-contributions shown in Table 1.3. The sub-contributions relating to Contribution II are shown in Table 1.4, namely studies S6 to S8. Agile and lean software development are introduced and systematically compared prior to the sub-contributions reported in S2 to S8 to provide the reader with sufficient background on the software development principles and practices associated with the paradigms (see study S1 in Table 1.2).

The first study S1 is a comparison of the lean and agile development paradigms (see Table 1.2). It is motivated by the need to understand the differences of the two approaches, as to the best of our knowledge, this has not been done systematically. The understanding facilitates the reuse of knowledge as lessons learned in regard to practices used in lean and agile are useful to understand both paradigms. Furthermore, understanding the differences allows to make informed decisions on how to complement the approaches. The comparison focuses on contrasting the goals, principles, practices, and processes that characterize the two paradigms. Thereby, the first study also provides the reader with a detailed introduction of the principles and practices that will be referred to in the remainder of the thesis. It also shows the differences between the two contributions of the thesis, the first one being related to agile software development (Contribution I), and the second one being primarily related to aspects that are unique to the lean software development paradigm (Contribution II).

Contribution I of the thesis is to understand the effect of the migration from plan-driven to agile software development. This contribution can be broken down into four sub-contributions, each addressing a research gap observed by studying the related work. An overview of the research gaps and individual chapters related to Contribution I are summarized in Table 1.3. Study S2 (Chapter 3) is the first step in understanding

Table 1.2: Comparison of Lean and Agile

Study	Research gap	Sub-Contribution
S1 (Chapter 2)	Need to understand difference between lean and agile aiding in generalizing benefits of lean and agile at the same time, and to show opportunities of complementing them for the identified differences.	Comparative analysis of lean and agile goals, principles, practices, and processes.

Table 1.3: Sub-Contributions of the Chapters Relating to the Migration from Plan-Driven to Agile Development (Contribution I)

Study	Research gap	Sub-Contributions
S2 (Chapter 3)	Few empirical studies on waterfall/ plan-driven, need for evidence supporting claims.	Qualitative study investigating bottlenecks/ unnecessary work/ avoidable rework in plan-driven and comparison with related work contributing empirical evidence.
S3 (Chapter 4)	Clear need for qualitative studies on agile to gain in-depth understanding, ans specifically with regard to the transition between development approaches and in large-scale context.	Capture change in perception of bottlenecks/ unnecessary work/ avoidable rework when migrating from plan-driven to agile development in a large-scale context.
S4 (Chapter 5)	Agile studies so far focus on small scale and single agile process (XP).	Qualitative study investigating bottlenecks/ unnecessary work/ avoidable rework in more detail after transition in large-scale context and comparison with related work.
S5 (Chapter 6)	No empirical analysis of lead-times in an agile context, open question in industry of how lead times are distributed between phases, the influence of number of impacted systems on lead-time, and unknown impact of size of requirements.	Quantitative analysis of lead-times with regard to distribution, impact, and size.

the effect of the migration from plan-driven to agile development by characterizing the baseline situation. The study in itself is motivated by the research gap that we were not able to identify studies with a sole focus on evaluating the benefits and drawbacks of plan-driven approaches empirically. To address this research gap and thus confirm or contradict the claims made in a wide range of software engineering books a qualitative study was conducted investigating the waterfall model as the prime representative of plan-driven approaches. Study S3 (Chapter 4) was motivated by the observation that most of the agile studies included in the systematic review by Dybå and Dingsøyr [8] were not explicitly investigating the migration, and hence little can be said about the effect of migrating in general, and in regard to large-scale software development in particular. In response we investigated what the effect of moving from the baseline situation presented in S2 to an agile process was. The focus here was on the change

of perception with regard to issues (e.g. bottlenecks) before and after the migration. Study S4 (Chapter 5) is different from Study S3 as it makes a more in-depth analysis of the new situation and compares the findings with those in the related work. In particular, the issues and advantages associated with agile software development were compared between the case study and empirical studies conducted by others. Finally, the lead-times of the new situation two years after the migration were investigated in Study S5 (Chapter 6) to capture the speed with which the requirements are delivered to the market. The goal of the company was to reduce the lead-time by setting realistic improvement targets. In order to set these targets it was necessary to understand how lead-times differ between phases, between requirements affecting single and multiple systems, and requirements with different sizes. If there is a difference it has to be taken into consideration. As this was an open question to the company it motivated an empirical study. In order to actually improve the lead-times lean provides a number of analysis tools and ideas that are focused on improving throughput and speed, leading to the Contribution II of the thesis, i.e. the proposal and evaluation of novel analysis approaches to improve the process with a focus of making it more lean and with that increase speed and throughput.

The second contribution (Contribution II) is the proposal and evaluation of lean software development tools to improve the agile processes in place. In Study S6 (Chapter 7) an approach for continuous improvement towards a lean process is presented. The study was motivated by the observation in related work that when introducing lean software development with a big bang strategy the introduction often failed. This observation has been made in lean manufacturing/ product development and in the software engineering context. Hence, the solution proposed (Software Process Improvement through Lean Measurement (SPI-LEAM Method) uses a measurement approach showing the absence of a lean process in different disciplines (e.g. development of software requirements, quality and test efficiency, maintenance, and so forth). The approach is based on the measurement of work in process (queues/inventories) and aims at achieving an integrated view on an abstract level to avoid sub-optimization. When detecting the absence in specific disciplines (e.g. maintenance) a more detailed look at the performance in this discipline should be possible. Approaches for the detailed analysis (i.e. drill-down view of each of the disciplines) are investigated in study S7 and S8. In study S7 (Chapter 8) a solution for the drill-down of main product development (i.e. inventory of requirements in the process life-cycle) is presented and evaluated. The goal in main product development is to achieve a smooth and continuous throughput. Whether this goal is achieved is not obvious with many requirements being developed in parallel in the large-scale context. Hence, we propose a visualization and measurement solution to be used in order to show the presence or absence of a continuous flow with high throughput. The approach has been evaluated empirically in a case study.

Table 1.4: Sub-Contributions of the Chapters Relating to the Implementation of Lean Software Development (Contribution II)

Study	Research gap	Sub-Contributions
S6 (Chapter 7)	Observed failure when introducing lean software development with a big bang approach.	Provide an analysis/ measurement approach allowing for continuous improvements towards a lean software process.
S7 (Chapter 8)	Goal of lean is smooth and continuous throughput which requires support in large-scale development with many requirements being developed in parallel.	Proposal of a visualization and measurement approach for the flow of main/ new product development and their industrial evaluation.
S8 (Chapter 9)	Isolated proposal of lean measures, no solution for holistic analysis, methods require much effort and many data points, no visualization to support management.	Proposal of a solution for lean maintenance driven by goals of lean software development and industrial evaluation.

Finally, Study S8 (Chapter 9) provides a solution for the drill-down of the maintenance discipline with regard to lean principles. As we observed some measures related to lean have been proposed, but they were proposed in isolation and thus do not provide a holistic picture of the maintenance flow. Hence, the proposed solution visualizes and measures the performance of lean maintenance and systematically integrates the individual measures.

1.4 Research Questions

Figure 1.2 provides an overview of the main research questions (QI.I, QI.II, and QII). The main research questions are linked to the research questions answered in the individual chapters, as illustrated by the arrows linking the main research questions and those stated for the chapters.

The first research contribution (Contribution I) is the evaluation of the migration from plan-driven development to agile development. For that contribution two main research questions are asked, one related to the usefulness of agile practices (QI.I) and one related to open issues being raised with the introduction of agile (QI.II). These main questions are linked to the research questions being answered in the papers. Question QI.I is linked to Chapters 3, 4, and 5 as those investigate questions that show the use-

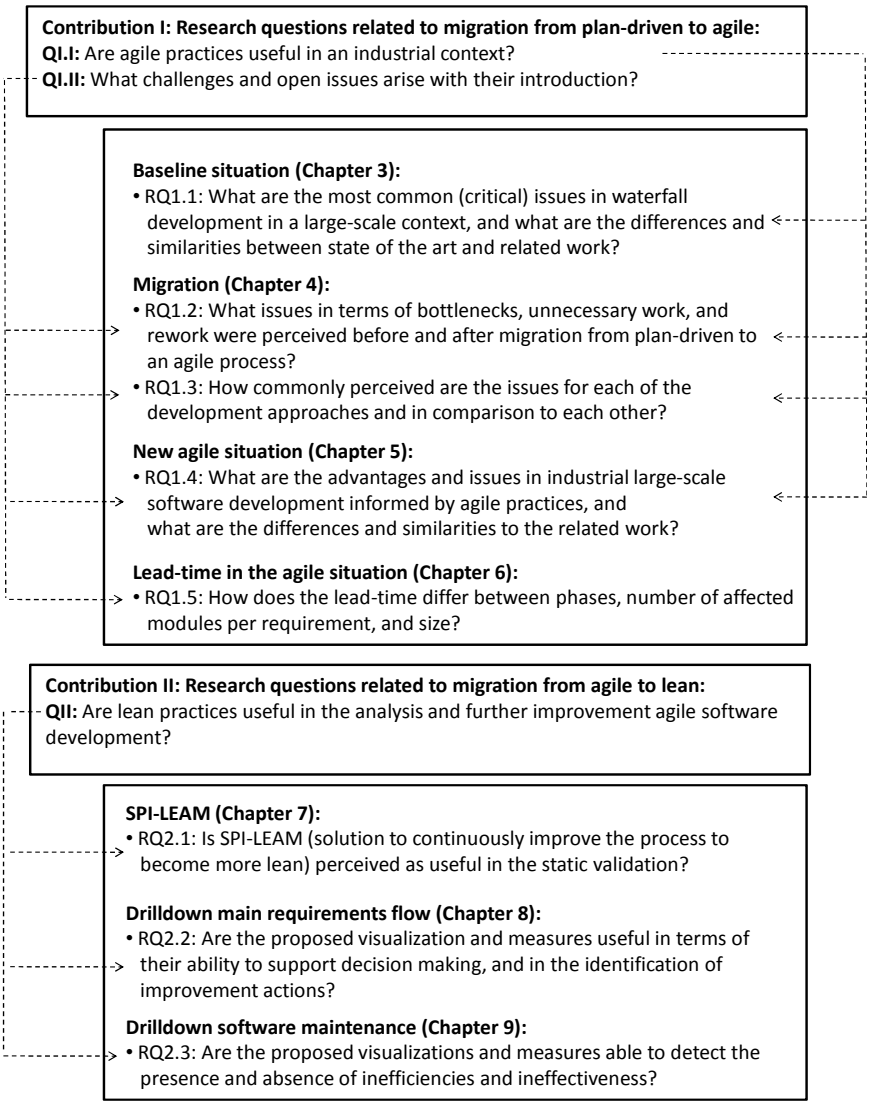


Figure 1.2: Research Questions

fulness by investigating whether an improvement is achieved from migrating from the baseline situation (plan-driven) to the new situation (agile). Question QI.II is looking for open issues that need to be addressed in order to leverage on the full benefits that agile software development seeks to achieve. Open issues are investigated in S3 by determining which of the issues in agile development are still perceived as very common. The questions in S4 have an explicit focus on the comparison of the issues identified in the case study with those reported in related work. Finally, study S5 looks at open issues from a lead-time perspective as the speed with which the company can react to customer needs was considered of high priority in the studied context.

The second research contribution (Contribution II) evaluates whether the addition of lean practices is useful in further improving the agile situation in place. The questions are very much related to the usefulness of the individual solutions proposed in the chapters. Study S6 evaluates the perceived usefulness of SPI-LEAM, an approach that evaluates how lean a software process is on a high level integrating different disciplines in one view (e.g. requirements flow, software maintenance, software testing, and so forth). The questions of Chapters 8 and 9 are related to the drill down to understand the behavior of the disciplines in more depth.

Chapter 2 connects Contribution I and II by answering the question of how the lean and agile development paradigms are different with regard to their goals, principles, practices, and processes.

1.5 Research Method

The choice of research method is motivated, followed by a description of the research questions, case and units of analysis, data collection and analysis, and validity threats.

1.5.1 Method Selection

Commonly used research methods in the software engineering context are controlled experiments [36], surveys [9], case studies [38], action research [33], and simulation [32].

Controlled Experiments: This research method is used to test theories in a controlled environment. For that purpose hypotheses are formulated regarding the cause-effect relationship between one or more independent variables and independent (outcome) variables. The experiment is conducted in a controlled environment, meaning that variables other than the independent variables should not have an effect on the outcome variables. For example, when testing the effect of two testing techniques on testing efficiency other factors such as experience should be controlled so they do not

affect the outcome. In practice it is often challenging to control the other factors, which makes it important to consider possible threats to validity when designing the experiment study. It is important to note that experiments with human subjects often need a sufficient number of participants carrying out a task for a longer period of time. In consequence, experiments are often conducted in a laboratory setting with students as subjects due to the limited availability of practitioners. The analysis of experiments is mainly based on statistical inference, e.g. by comparing whether there is a statistically significant difference with regard to the outcome variable for two different treatments.

Surveys: A survey studies a phenomena for a population by surveying a sample that is representative for that population. In order to collect the data from the sample questionnaires and interviews are often used. Online questionnaires are preferred as they allow to reach a larger sample and are less time consuming from the researchers point of view, and are thus more efficient than interviews with regard to the number of data points that could be collected (see, for example, [10] who received more than 3000 answers on their questionnaire). Having collected the data of the sample statistical inference is used to draw conclusions for the overall population.

Case Studies: Case studies are an in-depth investigation of a phenomena focusing on a specific case. The cases are objects of the real world studied in a natural setting, in software engineering this means they are real software organizations, software projects, software developers, etc. Case studies are conducted by defining the case to be studied, the units of analysis, as well as a data collection strategy. With regard to case studies a number of decisions have to be made. The first one being whether the study is of confirmative or exploratory nature [26]. A confirmative case study sets out with a proposition or hypotheses to be tested in the real world. In the case of the exploratory case study little knowledge about the phenomena in the real world is available and hence the study aims at identifying theories and propositions. When these are defined they can be confirmed in forthcoming confirmative case studies or by other research methods, such as experiments. Another decision to be made is whether a flexible or fixed design should be used [26]. A flexible design allows to change the design based on new information gathered while executing the case study, e.g. leading to a change of research questions or data sources. In a fixed design the researcher sticks with the initial design throughout the case study process. In comparison to controlled experiments there is much less control in an industrial case study, which means that confounding factors very likely play a role when making inferences, e.g. as is the case in this thesis where inferences were made about the effect of the transition between two development paradigms. The analysis of case studies ranges from a purely qualitative analysis where raw data from interviewees is categorized and coded to the exclusive use of statistical inference.

Simulation: Simulations are executable models of a real world phenomena (e.g.

software processes, software architecture) to study their behavior [32]. After building the simulation model complex real-world processes can be simulated by calibrating the model parameters based on empirical data, be it from existing publications or directly collected from software industry. Simulation is able to capture complex processes and systems to determine their performance, the calculation of the performance outcome being too complex for a human to calculate in a time-efficient manner. This is due to that software processes have complex looping mechanisms and the flow through the processes is determined by many different factors that occur with specific probabilities. Simulations have been used in many different application areas, e.g. release planning [1], requirements engineering [13], and quality assurance [7]. In these application areas simulation can be, for example, used for software productivity prediction [27] or to manage risk. Simulation as a tool for empirical research can be used to test a new solution in a "what-if" scenario, such as introducing a new testing technique into a process and determine the impact on multiple process performance parameters.

Action Research: In action research the goal is to introduce an intervention in a real-world setting and then observe what the affect of the intervention is. The researcher is actively involved in introducing the intervention and making the observations [26], in fact the researcher takes an active part in the organization (e.g. by participating in a development team affected by the intervention introduced). The steps of action research are planning of the intervention and the collection of the data to capture the intervention effect, the implementation of the actual intervention, and the collection of data and their interpretation [33]. As pointed out by Martella et al. [18] much can be learned by continuously observing the effect of a change after inducing it. However, as the researcher is actively involved in the team work action research is an effort intensive approach from the researcher's point of view.

Motivation for Choice of Research Method: The research method of choice for this thesis was case study. The motivation for the choice is given by a direct comparison with the other methods.

- *Case study vs. experiment:* As mentioned earlier, the contribution of this thesis is the investigation of the implementation of lean and agile practices in a real-world setting. As processes are concerned and the goal was to focus on large-scale software development with hundreds of people involved it is not possible to replicate such a solution in a lab environment with students.

- *Case study vs. survey:* A survey is classified as research in the large to get an overall overview of a phenomena with regard to a population, hence being referred to as research in the large [26], or research in the breadth. As in this study the aim was to gain an in-depth understanding research in the breadth is not an option. For example, asking too many detailed and free-text questions in a

survey would likely result in respondents not completing the survey. Hence, case study is more suitable when it is agreed with a company that people are available for detailed interviews allowing for the in-depth understanding.

- *Case study vs. simulation:* Simulation primarily focuses on measurable characteristics of a process. As in-depth understanding requires a qualitative approach for research case study is given preference over simulation.

- *Case study vs. action research:* Action research is very much based on planning an action or intervention and observing its effect. In the first contribution of the thesis this was not possible as the action has been taken (i.e. the migration from plan-driven to agile). For the second contribution (introduction of novel approaches to lean software engineering) action research would have been a suitable alternative for the research goals, but was not chosen due to the effort connected with that, specifically as the intervention was done throughout the overall development site at the studied company. Thus, instead of taking active part in all implementations the researcher acted more as an observer.

Overall, the comparison shows that case study was considered the most suitable research method in order to achieve the research contributions. The case studies are confirmative as propositions could either be defined based on literature (studies S2 to S5), or the case studies seek to confirm the usefulness of solutions (S6 to S8). The case study design was fixed and agreed on with the company.

It is also important to distinguish between static and dynamic validation. Static validation means to present the solution to practitioners and then incorporate their feedback. This step is important to receive early feedback and to get a buy-in for the implementation of the solution. Dynamic validation is the evaluation of the actual usage of the solution in an industrial setting. The actual use (dynamic validation) is conducted through case study research. If no dynamic validation was used yet, but we presented the solution to the practitioners to receive early feedback, we refer to the research method as static validation.

Within the case studies a number of sub-methods have been used, namely interviews (S2 to S4) and workshops (S6 and S7) for data collection, and grounded theory (S2 to S4) as well as statistical analysis (S3, S6 to S8).

Interviews: Interviews are conversations guided by an interview protocol and are considered one of the most important resources for data when conducting case studies [38]. The interview protocol can vary in the degree of structure, ranging from very structured (interviewee has to stick with research questions) over semi-structured (interviewee has a guide, but can change the course of the interview to follow interesting directions) to unstructured (rough definitions of topics to be covered). In this case

study we used semi-structured interviews to allow for some flexibility in the conversation. Unstructured interviews were not considered as interviews with some structures seem to be the most efficient way of eliciting information [6].

Workshops: In a workshop a group of people is working together to solve a task. A moderator is leading the workshop to guide the participants in achieving the task. Workshops have been used in the studies where a new solution was provided to the company to gather feedback. Workshops have been chosen to allow for an open discussion and reflection on the introduced solutions considering different roles. The advantage of workshops is that they are very efficient in collecting data from several perspectives in only one occasion. The disadvantage is that workshops require several people to be available at the same time on one occasion, which can be a challenge when conducting research with industry. Hence, workshops were used in the later phases of the research (implementation of lean in studies S6 to S8) as it was easier to get the right people at the same time. That is, the solution was under implementation at the studied company with management support, assigning people to reflect and support the improvement of the solution implementation.

Grounded Theory: The goal of grounded theory is to develop theories [26]. In this research the goal was not to develop theories. However, grounded theory provides valuable tools to analyze an overwhelming amount of qualitative data [26]. The following concepts from grounded theory have been applied to this research: reduction of data through coding, display of data (use of matrices and tables), and documenting relationships in mind-maps with narrative descriptions of the branches of the maps. The detailed descriptions of the analysis process can be found in the chapters describing studies S2 to S4.

Statistical Analysis: Descriptive statistics are also a means of reducing the amount of data, and to visualize quantitative data in order to aid analysis [26]. Different kinds of diagrams have been used in the studies, such as box-plots (S5 and S8), bar charts (S7), etc. In study S5 we were interested in establishing relationships between different variables. Hence, correlation analysis and regression was used to determine whether one variable accounts for the variance in another variable.

1.5.2 Case and Units of Analysis

The case being studied is Ericsson AB located in Sweden. The company is developing software in the telecommunication and multimedia domain. The market to which the products are delivered can be characterized as highly dynamic and customized. Customized means that customers often ask the delivered product to be adapted to their specific needs after release. The company is ISO 9001:2000 certified. As the company is delivering to a dynamic market the type of development is market-driven. That is,

the customers to which products are delivered are not all known beforehand. Instead, the product is developed for many potential customers operating on the market. In contrast, in bespoke development the customer and the users are known beforehand and thus the main customers with that users are very clear when developing the software. Additional details on the context elements relevant for the individual studies are presented within the chapters.

The units of analysis were the systems developed at the case company. Two major systems are developed at the development site that was investigated in this thesis. Each of the systems is of very large scale and thus the systems are broken down into nodes (also referred to as sub-systems). Different nodes have been studied throughout the chapters of the thesis and are labeled and discussed within the chapters. The number of sub-systems studied in each study are also stated in Table 1.5.

1.5.3 Data Collection and Analysis

Table 1.5 provides an overview of the data collection approaches and the evaluation criteria for determining the usefulness of lean and agile development. In addition, the research methods used and the units of analysis are summarized. For the collection in the industry studies we used interviews, quantitative data collected at the company, process documentation, and workshops.

Interviews: Chapters 3, 4, and 5 are based on a large-scale industry case study and the data was primarily collected from a total of 33 interviews. The interviews were semi-structured meaning that the interviewee used an interview guide, but was allowed to depart from the guide and ask follow-up questions on interesting answers raised during the interviews [26]. In that way the semi-structured interviews provide some flexibility to the interviewee. Questions regarding the experience of the interviewees were closed questions while the actual interview only contained open-ended questions. Interviews have been chosen as they allow to gain an in-depth understanding of the situation at the company. A systematic review in the context of requirements elicitation, requirements elicitation aiming at gaining an in-depth understanding of the customers' needs regarding a software system to be developed, showed that interviews with some structure seem to be the most effective way to elicit information [6], which is supporting our choice of method.

Quantitative data: In addition to the qualitative data collected in Chapters 3, 4, and 5 quantitative data on the effect of the migration from plan-driven to an agile development approach has been used. The data source for that was data collected at the company, which was closed source. That means, the original data sources were not visible to the researchers and with that there was little control on the data collection and with that constitutes a risk to validity. The collected data in Chapters 6, 7, and

Table 1.5: Data Analysis and Evaluation Criteria

Study	Research Method	Units of Analysis	Data Collection	Evaluation Criteria
S1 (Chapter 2)	Literature Survey	Generic descriptions of development approaches for lean and agile software development.	Reference databases	Pairwise comparison of goals, principles, practices, and processes to identify similarities and differences.
S2 (Chapter 3)	Case Study	Three sub-systems at the case company	Interviews, quantitative data collected at the company, process documentation	Commonality of perception of bottlenecks / unnecessary work / avoidable rework in the plan-driven (baseline) situation.
S3 (Chapter 4)	Case Study	Three sub-systems at the case company	Interviews, quantitative data collected at the company, process documentation	Commonality of perception of bottlenecks / unnecessary work / avoidable rework before and after migration; Additional data source (closed company data) as a support for analyzing the change.
S4 (Chapter 5)	Case Study	Three sub-systems at the case company	Interviews, process documentation	Commonality of perception of bottlenecks / unnecessary work / avoidable rework in the agile situation
S5 (Chapter 6)	Case Study	All sub-systems developed at development site studied at case company.	Company-proprietary system developed for the lean solution presented in this paper.	Descriptive statistics (box-plots) and an analysis of whether variables (size, system impact, phase) account for the variance in lead-time.
S6 (Chapter 7)	Static Validation	One sub-system at case company to exemplify approach for one inventory	Company-proprietary system developed for the lean solution presented in this paper, Workshops	Showing applicability on an industrial example and practitioner feedback
S7 (Chapter 8)	Case Study	Nine of the sub-systems developed at development site studied at case company, illustration of the approach on a sub-set of these systems	Company-proprietary system developed for the lean solution presented in this paper, Workshops	Practitioner feedback where practitioners identify decisions in which they believe the approach is an useful support. Ability of the approach in aiding practitioners in using the measures to identify improvements.
S8 (Chapter 9)	Case Study	Maintenance process for one sub-system at studied company.	Company proprietary tool that was already in place for software maintenance, Workshops	Ability of the solution to show presence / absence of inefficiencies or ineffectiveness.

8 was based on a company-proprietary tool constructed by the author and a colleague at Ericsson. Hence, in that case the data was available to the researcher having full access, and thus was able to conduct quality checks on the data to assure completeness and consistency. The study presented in study S8 also made use of a company proprietary tool for defect tracking with full access to the researcher. Therefore, the data in study S8 was also under the researcher's control.

Process documentation: Process documentation played an important role in the early studies of the paper presented in Chapters 3 to 5 as they allowed to gain knowledge about the terminology used in the company. Furthermore, a basic understanding of the processes and ways of working was achieved, which was useful when conducting the interviews. That is, the study of company documentation made the communication between the researcher and practitioners much easier and hence allowed to focus the interviews on the actual issues rather than clarifying terminology.

Practitioner Workshops: Practitioner workshops were used to gather feedback on the proposed solutions as well as the researcher's interpretation of the research results. In general workshops were organized by providing an introduction to the theme(s) of the workshop and then the themes were openly discussed in the workshops. Some workshops also provided tasks to the workshop participants, such as writing notes and presenting them to the audience (see e.g. study S7). How individual workshops were organized and who participated in them is presented in the individual chapters. Overall, we found workshops to be efficient in gathering feedback as they allow to openly discuss the same theme from different perspectives and roles at a single occasion.

Regarding the analysis different evaluation criteria have been used. Studies 2, 3, and 4 used the commonality of responses across different roles in the development life-cycle and across sub-systems for evaluation purposes. If an issue, for example, is not perceived as common in the new way of working (agile) in comparison to the old way of working (plan-driven) then this indicates an improvement. The evaluation of the lean approaches was primarily based on the feedback of the practitioners, and their reflections when using them. The evaluation criteria for each study are summarized in the very right column in Table 1.5.

1.5.4 Validity Threats

Four types of validity threats are commonly discussed in empirical studies such as experiments [36] and case studies [38, 29]. Construct validity is concerned with choosing and collecting the right measures for the concept being studied. Internal validity is concerned with the ability to establish a casual relationship statistically or the ability to make inferences. External validity is about the ability to generalize the findings of the study. Finally, conclusion validity is concerned with the ability of replicating the study

and obtaining the same results. Table 1.6 provides an overview of the validity threats observed throughout the case study, stating and describing the threats with a reference to the concerned studies.

A threat to validity in studies S2 to S4 is the unbiased selection of people for the interviews. Possible biases could be that only interviewees are selected that are positive towards one of the models. To avoid this threat the interviewees were randomly selected to collect the data for studies S2 to S4.

Reactive bias is concerned with the presence of the researcher affecting the outcome of the study. This threat is only relevant when people are involved in either interviews (S2 to S4) or workshops (S6 and S7) that might perceive the researcher as external and hence behave differently in comparison to being only with their peers. As the researcher was partly employed at the company he was perceived as internal, which mitigates the threat of reactive bias.

Correct data is a validity threat throughout all studies (S2 to S8) as they are all of empirical nature. In the case of the interviews the correct data was assured by taping the interviews (S2 to S4). In S2 and S3 we also used closed data sources of data provided by the company, which remains a validity threat as the original data source cannot be examined. Therefore, the quantitative data in S2 and S3 only serves as an additional data source to the qualitative data collected in the interviews, but should not be used as an indicator of how much quantitative improvement could be achieved when migrating from one development paradigm to the other. The correctness of the data in the workshops (S6 and S7) was assured by comparing notes with a fellow colleague at the company who also documented the outcome of meetings and workshops. The quantitative data collected in studies S5 to S7 were based on a company proprietary system specifically designed for the purpose of collecting data related to the lean principles introduced in this thesis and hence were available to the researchers. The software maintenance study in S8 is based on a defect tracking system already in place at the company, which was also fully accessible to the researcher. Hence, the main threat to correct data remaining in this thesis is the closed data source concerning studies S2 and S3.

The background and goals of the researcher could bias the interpretation by the researcher. In order to reduce this threat to validity in studies S2 to S4 the analysis steps for the interviews were reproduced in a workshop with practitioners and the authors present, showing that all agreed with the analysis done. The interpretations of the quantitative data (S5 to S8) and the data from the workshops (S6 and S7) was discussed with colleagues at the company. In addition, all studies have been reviewed by a colleague of the company who was involved in the introduction of the approaches, and also was present in the workshops and meetings. The colleague confirmed the findings and approved the studies, hence being an important quality control with regard to bias.

Another threat is that one specific company and thus company specific processes are studied. Therefore, the context and processes have been carefully described to aid other researchers and practitioners in the generalization of the results. In order to identify the relevant context information for the studies included in this thesis we used a checklist of context elements introduced in Petersen and Wohlin [22], except for study S2 and S4 as the checklist has been developed after having the possibility to make changes to these studies.

Confounding factors are important when making inferences about a root-cause relationship which is the case for study S3 in which inferences are made about the change due to the migration from plan-driven to agile development. The confounding factors cannot be ruled out as the study was conducted in an uncontrolled industrial environment. In order to address the validity threat the most obvious factors were ruled out and a person involved in the measurement collection was asked about confounding factors, saying that at least partially the change could be attributed to the migration from plan-driven to agile development.

The ability to make inferences in study S3 could be negatively influenced if the instrument for data collection (interview) is not able to capture the change due to the migration. This threat has been reduced by explicitly asking for the situation before and after migration, and has been documented through citing the statements made by the interviewees which explicitly contained the inferences made in the presentation of the case study. In study S5 statistical analysis was used to determine whether lead-time varies with regard to phase, system impact, and size. The inference to a population is limited as the data is not drawn from a random sample. Hence, the context was carefully described as companies in a similar context (large-scale development, parallel system development, and incremental deliveries to system test) are more likely to make similar observations.

Overall, the analysis of the threats to validity shows that throughout the studies included in the thesis actions have been taken to minimize the threats. Further details about the threats in the context of the individual studies is provided within the chapters.

1.6 Results

Table 1.7 summarizes the results of the individual studies. The table shows the sub-contributions of the individual study/chapter and shortly provides a description of the main result. The results of studies S2 to S5 are linked to Contribution I and the results of studies S6 to S8 to Contribution II. The descriptions of the study results provide answers to the research questions linked to the individual chapters (see Figure 1.2).

In the very first study (S1) the lean and agile development paradigms evaluated

Table 1.6: Validity Threats Observed in Empirical Studies at Case Company

Threat	Description	Concerned Studies
Unbiased selection	Researcher not biased in selecting people in interviews	S2, S3, S4
Reactive bias	Presence of researcher influences outcome	S2, S3, S4, S6, S7
Correct data	Complete and accurate data	S2, S3, S4, S5, S6, S7, S8
Researcher bias	The interpretation of the data is biased by the researcher	S2, S3, S4, S5, S6, S7, S8
One company	Company specific context and processes	S2, S3, S4, S5, S6, S7, S8
Confounding factors	Other factors affecting the outcome that are not controlled	S3
Inference	Ability to make an inference regarding improvements	S3, S5

in the thesis were compared with each other to make the differences and similarities between the two development paradigms explicit. Lean and agile are similar in goals. Hence, some principles are similar as well related to people management leadership, technical quality of the product, and release of the product. In addition, the paradigms complement each other. For example, lean provides concrete examples for overhead in the development process to become lightweight and agile. Unique to lean is the principle of seeing the whole of the software development process, e.g. using systems thinking, value stream maps, and other lean tools. With regard to quality assurance and software release the same practices are considered (e.g. test driven development). Unique to agile are the principles agile on-site customer, coding standards, team-code owner ship, planning game, 40 hour week, and stand-up meetings.

Study S2 analyzes the baseline (plan-driven) situation before the migration to agile development. The issues identified through interviews were classified into groups based on the commonality of responses that were mentioned by the interviewees across different systems studied, and across different roles. Four classes (A to D) were defined by setting thresholds with regard to the number of responses, the thresholds being used as a means to structure the results with regard to commonality of the issues. The study provides the following answer to research question *RQ1.1*: The most common issues (A and B) were related to requirements and verification. In the requirements phase many requirements were discarded as they became obsolete due to long lead-times in the plan-driven process. That is, everything has to be finished before it can be delivered,

Table 1.7: Overview of Results

Study	Sub-Contribution	Result
S1 (Chapter 2)	Comparison lean and agile	Main difference is that lean focuses on the end to end analysis of the development flow to get a complete picture of the behavior of development. All other principles can be found to some degree in both paradigms; Principles unique to agile and principles unique to lean could be identified.
S2 (Chapter 3)	Migration plan-driven to agile	Most common issues related to requirements (large amount of discarded requirements) and verification (reduced test coverage, increased amount of faults due to late testing, faults found late hard to fix), hence not suitable in large-scale development.
S3 (Chapter 4)		Problems originally perceived in waterfall development are less commonly perceived in agile development indicating improvement. Measures support this outcome. Some open issues specific to agile remain.
S4 (Chapter 5)		High overlap in advantages identified for benefits in smaller scale (literature) and large scale. Few issues have been mentioned in empirical studies as many issues related to scaling agile. Hard to compare studies in agile as context descriptions need improvement.
S5 (Chapter 6)		No significant difference between phases. System impact does not lead to difference in lead-time. Size increases lead-time in implementation phase.
S6 (Chapter 7)	Evaluation of lean practices	Practitioners agreed with the assumptions of the approach and they found illustration of measurements easy to understand/ use. Overall, the approach was perceived as useful.
S7 (Chapter 8)		Practitioners identified short-term and long-term decisions in which the approach is useful. In addition, the measures led the practitioners to identify a number of improvement proposals.
S8 (Chapter 9)		Approach was able to show presence and absence of inefficiencies and ineffectiveness in the maintenance process.

which takes some time and during that time the needs of the customers change. The most common issues with regard to verification were test coverage, amount of faults

increase with late testing, and faults found in the process are hard to fix. All issues classified as A and B were mentioned in literature supporting the generalizability of the results. Hence, overall the study confirms the findings in literature, while adding some new, but less commonly perceived issues, as described in S2. The conclusion was that the waterfall approach was not suitable in the studied context. Hence, the study supports the decision made by the company to change towards a more agile process.

Study S3 compares the plan-driven situation (from S2) with the situation after migration to agile development. In research question *RQ1.2* we asked for the perceived issues before and after the migration. The most common issues before the migration are presented in S2 and S3. The most common issues remaining after the migration for the agile situation were:

- Test cycle planning prolong lead-times in case of a rejection of an increment by the test, or if an increment is delivered too late (increment has to wait for completion of the following test cycle);
- Reduction of test coverage due to lack of independent verification and validation and short projects putting time pressure on teams; (3) people concerned with the release are not involved early in the process;
- Project management overhead due to high number of teams (much coordination and communication).

The benefits of the migration were:

- More stable requirements led to less rework and reduced waste in the requirements phase; (2) estimations are more precise;
- Early fault detection allows to get feedback from test early on;
- The lead-time for testing was reduced;
- Moving people together reduced the amount of documentation needed as direct communication could replace documentation.

Given the results we can answer research question *RQ1.3* by saying that the problems reported in S2 with regard to requirements and verification are not perceived as that common anymore after the change, the exact ratings of the commonalities being stated within the Chapters presenting S2 and S3. The measurements, which were based on a closed data source and were used for the purpose of triangulation, supported the qualitative results. Overall this is already a positive result as the introduction of the new practices had been done quite recently at the time the case study was conducted.

In study S4 an in-depth investigation of the situation after migration is conducted. For that purpose the thresholds with regard of how often an issue (or advantage) has been mentioned is reduced to gain a more in-depth understanding of the new situation, the results of the study answering research question *RQ1.4*. Additional issues to those identified in study S3 were:

- Handover from requirements to implementation takes time due to complex decision processes;
- The requirements priority list used is essential in for company's model to work and is hard to create and maintain;
- Design has free capacity due to long lead-times as in requirements engineering complex decision making takes place (though as shown in previous study this is less severe);
- Too much documentation in testing, but less severe;
- Many releases on the market mean many different versions of the system that needs to be supported which might increase maintenance effort;
- Configuration management requires high effort to coordinate the high number of internal releases;
- The development of the configuration environment to select features for customizing solutions takes a long time;
- Product packaging effort is increased as it is still viewed from a technical point of view, but not from a commercial point of view;
- Dependencies rooted in implementation details are hard to identify and not covered in the anatomy plan.

Additional benefits were:

- Time of testers used more efficiently as testing and design can be easily parallelized due to short ways of communication;
- Higher transparency of who is responsible for developing increments, this generating incentives for delivering quality.

With regard to the comparison with related work we found that existing empirical studies and study S4 agreed on the advantages that could be achieved with agile software

development. However, some new issues have been identified that need to be addressed to fully leverage on the benefits of agile software development.

Study S5 set out to investigate effects of different attributes related to requirements on lead-time (difference between phases, difference with regard to the number of systems affected, and difference with regard to size). The following answers for research question *RQ1.5* were obtained:

- No significant difference between lead-times of phases could be established based on statistical tests (i.e. no specific time intensive activity);
- No difference between requirements affecting one or multiple systems, which was considered a surprising result;
- Large requirements have a tendency of increased lead-time in the implementation phase, but not in other phases.

In order to further improve the lead-time situation observed in study S5 lean tools are proposed to identify and remove wastes in the development process, which leads to the results of Contribution II of this thesis (see Table 1.7).

Study S6 presents a novel approach called software process improvement through the lean measurement (SPI-LEAM) method. SPI-LEAM evaluates how well an overall software process performs considering the work in process in comparison to capacity. The approach facilitates a combined analysis of different dimensions of the software process (such as main product development, software testing, software maintenance, and so forth). The dynamic validation conducted with practitioners showed that the overall approach was perceived positively, and that the practitioners agreed with the underlying assumptions of the approach, e.g. that the work-load should be below capacity as this allows for a continuous flow of development (research question *RQ2.1*). At the same time being below capacity provides flexibility in case of required maintenance or other high priority tasks emerging.

The previous study (S6) analyzed the overall process. When undesired behavior is discovered (e.g. in particular dimension of the process) it might be necessary to take a closer look at these dimensions. Study S7 proposes a visualization of the flow of requirements through development in combination with a set of measures. The measures allow for the discovery of bottlenecks, discontinuous work flow, and the discovery of waste in forms of discarded requirements. The approach was evaluated by practitioners using the approach and then reflecting upon it. The findings were that:

- Requirements prioritization is supported;
- The measures aid in allocating staff;

- The measures provide transparency for teams and project managers of what work is to be done in the future and what has been completed;
- Software process improvement drivers can use the measures as indicators to identify problems and achieve improvements from a long-term perspective.

In addition we evaluated what improvement actions practitioners identified based on the measurements. The identified improvements were:

- An increased focus on continuous development by limiting the allowed number of requirements in inventories;
- Earlier and more frequent integration and system testing of the software system to increase quality.

The case study showed that the visualization and measures are perceived as valuable from an industrial perspective and that the practitioners could identify improvement actions (research question *RQ2.2*).

Study S8 proposed lean measures and visualizations specifically tailored towards the needs of software maintenance. Individual visualizations and measures were proposed for the inflow of maintenance requests, flow of requests for maintenance tasks throughout the maintenance process, lead-times, and work-load. The approach has been applied to an industrial case for one system and it was demonstrated that the presence or absence of inefficiencies and ineffectiveness in the maintenance process could be identified (research question *RQ2.3*). The major improvement potential identified was the reduction of waiting times and the need for a more continuous work-flow in some phases of the maintenance process.

1.7 Synthesis

The synthesis draws together the obtained results in order to provide answers to the main research questions asked in this thesis.

- *QI.I: Are agile practices useful in an industrial context?* In the study investigating the baseline (plan-driven development) it was shown that many critical issues were identified, making the need for more agility and flexibility explicit (S2). The need is further strengthened by the fact that the company decided to move towards agile practices across all its development sites. With the introduction of agile practices we have shown that the practitioners perceived the migration positively, mentioning multiple advantages. At the same time major issues that were

commonly perceived for the plan-driven approach are less commonly perceived in the new process (S3). Finally, study S4 adds further advantages when looking at the process in detail, as presented in the previous section (Results). The quantitative data, though facing validity threats of a closed data source, supports the qualitative results, which further strengthens the qualitative result from the interviews. In addition it has to be considered that the migration was done quite recently and for that considerable improvements have already been achieved. Hence, the answer to the first research question is that agile practices appear to be useful in a large-scale industrial context.

- *QI.II: What challenges and open issues arise with the introduction of agile practices?* Though, the result showed improvement there is still potential to further improve the new situation after migration as open issues have been identified. The most critical open issues were reported in study S3 investigating the effect of the migration, namely long lead-times due to test cycle planning, reduction of test coverage, release planning involved too late in the process, and overhead due to coordination needs. Further open issues have been identified in study S4. In study S5 lead-times were analyzed showing no difference in lead-times between phases, and no differences between requirements affecting one system and several systems. Based on the lead-time analysis the company seeks to achieve further improvements. This leads to the next question concerning lean practices, which have shown major improvements in lead-time reduction and value creation for customers in the manufacturing and product development domain.

- *QII: Are lean practices useful in the analysis and further improvement of agile software development?* The lean approaches proposed in this thesis have been evaluated using practitioner feedback and the reflection of the practitioners when using them. The approaches were perceived useful, this is true for the SPI-LEAM solution, but also for the solutions of analyzing main product development and the maintenance process. SPI-LEAM is able to integrate the different dimensions of the process. However, the individual solutions presented in studies S6 and S8 were also perceived as valuable stand-alone, meaning that SPI-LEAM (S6) is not a pre-requisite for the solutions presented in S7 and S8 to be useful. In addition, the usefulness of the approaches is supported by the fact that the company is adopting them, and in particular is using the solution presented in S7 continuously. Now that the lean practices have been used for almost a year it can also be said that they are a driver for improvements, e.g. the implementation of Kanban principles (see Chapter 2 for a description) and other improvements. In consequence, the answer to this research question is that lean practices are useful in further improving agile processes.

1.8 Conclusions

In this thesis two contributions are made, namely the usefulness of the implementation of agile software development (Contribution I) and the added benefit of using lean practices on top of that (Contribution II).

In order to determine the usefulness of agile practices (Contribution I) a series of four studies was used. In the first study the baseline situation was analyzed (i.e. the plan-driven situation). Thereafter, the effect of moving from the plan-driven to an agile process was investigated. Finally, an in-depth analysis of the new situation was described. In addition an analysis of lead-times in the situation after the migration has been conducted.

With regard to the lean practices the goal was to determine if they have some added value to agile practices (Contribution II). Therefore, novel approaches strongly inspired by lean thinking as presented in the context of lean manufacturing and product development were proposed. The first approach (referred to as software process improvement through the lean measurement method, short SPI-LEAM) analyzes the capacity of the process in comparison to the workload in order to avoid overload situations and with that realize a smooth flow of development. The solution is multi-dimensional in that it integrates different dimensions of the process (such as main product development, software maintenance, software testing, and so forth). When observing undesired behavior with SPI-LEAM in specific dimensions a drill-down analysis should be possible to understand the problem in more detail. Therefore, lean approaches for the main product development flow of requirements and for software maintenance have been proposed.

To build a bridge between the agile and lean software development a chapter on the difference between these two development approaches has been included.

The evaluation of the agile practices has been done through a series of three industrial case studies. All case studies have been conducted at Ericsson AB. The outcome of the case studies showed that the migration from plan-driven development to agile development is beneficial as improvements in several areas (primarily requirements engineering, and verification) could be achieved. Furthermore, open issues and challenges have been identified that have to be addressed in order to fully leverage on the benefits that agile practices can provide.

The evaluation of the lean practices showed that they were generally perceived as useful by the practitioners. In addition, the lean practices aided the practitioners in identifying improvements, and they also identified concrete decisions in which the lean practices provide valuable support. The feedback by the practitioners was based on their experience when using the approaches.

With regard to future work further studies are needed on agile software development in general, and with a focus on large-scale software development in particular.

Furthermore, more mature agile implementations need to be studied to understand the long-term effects in terms of benefits and challenges that come with their introduction. With regard to lean future work is needed to evaluate the lean practices and solutions proposed in this thesis in different contexts. As with agile the long-term effect of using lean practices needs to be investigated. Only a sub-set of lean tools has been proposed in this thesis, but there are others as well (e.g. value-stream maps). Hence, a comparison of different lean tools would be beneficial to support in the selection of the most suitable approaches.

1.9 References

[1] Ahmed Al-Emran, Puneet Kapur, Dietmar Pfahl, and Günther Ruhe. Simulating worst case scenarios and analyzing their combined effect in operational release planning. In *Proceedings of the International Conference on Software Process (ICSP 2008)*, pages 269–281, 2008.

[2] Sebastian Barney, Aybüke Aurum, and Claes Wohlin. Software product quality: ensuring a common goals. In *Proceedings of the International Conference on Software Process (ICSP 2009)*, pages 256-267, 2009.

[3] Kent Beck. *Extreme Programming explained: embrace change*. Addison-Wesley, Reading, Mass., 2000.

[4] Stefan Biffl, Aybüke Aurum, Barry Boehm, Hakan Erdogmus, Paul Grünbacher. *Value-based software engineering*. Springer, Heidelberg, 2005.

[5] Christian D. Buckley, Darran W. Pusipher, and Kandell Scott. *Implementing IBM rational clearQuest: an end-to-end deployment guide*. IBM Press, 2000.

[6] Alan M. Davis, Óscar Dieste Tubío, Ann M. Hickey, Natalia Juristo Juzgado, and Ana María Moreno. Effectiveness of requirements elicitation techniques: empirical results derived from a systematic review. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE 2006)*, pages 176–185, 2006.

[7] Paolo Donzelli and Giuseppe Iazeolla. A hybrid software process simulation model. *Software Process: Improvement and Practice*, 6(2):97–109, 2001.

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

[9] Floyd J. Jr. Fowler. *Improving survey questions: design and evaluation*. Sage Publications, Thousand Oaks, California, 1995.

[10] Tony Gorschek, Ewan Tempero, and Lefteris Angelis. A large-scale empirical study of practitioners' use of object-oriented concepts. In *Proceedings of the Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, 2010.

[11] Tony Gorschek and Claes Wohlin. Requirements abstraction model. *Requir. Eng.*, 11(1):79–101, 2006.

[12] Michael Hirsch. Moving from a plan driven culture to agile development. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, page 38, 2005.

[13] Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, and Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3):323–332, 2001.

[14] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements engineering*. Springer, London, 2005.

[15] George Stalk Jr. Time - the next source of competitive advantage. *Harvard Business Review*, 66(4), 1988.

[16] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley, Boston, 2004.

[17] Craig Larman. *Agile and iterative Development: a manager's guide*. Pearson Education, 2003.

[18] Ronald C. Martella, Ronald Nelson, and Nancy E. Marchand-Martella. *Research methods : learning to become a critical research consumer*. Allyn & Bacon, Boston, 1999.

[19] Peter Middleton. Lean software development: two case studies. *Software Quality Journal*, 9(4):241–252, 2001.

[20] Peter Middleton, Amy Flaxel, and Ammon Cookson. Lean software management case study: Timberline inc. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, pages 1–9, 2005.

[21] James M. Morgan and Jeffrey K. Liker. *The Toyota product development system: integrating people, process, and technology*. Productivity Press, New York, 2006.

[22] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, pages 401–404, 2009.

[23] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, Boston, 2003.

[24] Mary Poppendieck and Tom Poppendieck. *Implementing lean software development: from concept to cash*. Addison-Wesley, 2007.

[25] Mary Poppendieck and Tom Poppendieck. *Leading lean software development: results are not the point*. Addison-Wesley, Upper Saddle River, NJ, 2010.

[26] Colin Robson. *Real world research: a resource for social scientists and practitioner-researchers*. Blackwell, Oxford, 2002.

[27] Jorge L. Romeu. A simulation approach for the analysis and forecast of software productivity. *Computers and Industrial Engineering*, 9(2):165–174, 1985.

[28] Walter Royce. Managing the development of large software systems: Concepts and techniques. In *Proc. IEEE WESCOM*. IEEE Computer Society Press, 1970.

[29] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[30] Viktor Schuppan and Winfried Rußwurm. A CMM-based evaluation of the V-model 97. In *Proceedings of the 7th European Workshop on Software Process Technology (EWSPT 2000)*, pages 69–83, 2000.

[31] Ken Schwaber. *Agile project management with Scrum*. Microsoft Press, Redmond, Wash., 2004.

[32] Forrest Shull, Janice. Singer, and Dag I. K. Sjøberg. *Guide to advanced empirical software engineering*. Springer-Verlag London Limited, London, 2008.

[33] Bridget Somekh. *Action research: a methodology for change and development*. Open University Press, Maidenhead, 2006.

[34] Glen L. Urban, Theresa Carter, Steven Gaskin, and Zofia Mucha. Market share rewards to pioneering brands: an empirical analysis and strategic implications. *Management Science*, 32(6):645–659, 1986.

[35] Brian White. *Software configuration management strategies and Rational ClearCase: a practical introduction*. Addison-Wesley, Harlow, 2000.

[36] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in software engineering: an introduction (international series in software engineering)*. Springer, 2000.

[37] James P. Womack, Daniel T. Jones, and Daniel Roos. *The machine that changed the world: how lean production revolutionized the global car wars*. Simon & Schuster, London, 2007.

[38] Robert K. Yin. *Case study research: design and methods*. Sage Publications, 3 ed. edition, 2003.

Chapter 2

Is Lean Agile and Agile Lean? A Comparison between Two Software Development Paradigms

Kai Petersen
To appear in Book Modern Software Engineering Concepts and Practices: Advanced Approaches, IGI Global

2.1 Introduction

The nature of software development has changed in recent years. Today, software is included in a vast amount of products, such as cars, mobile phones, entertainment and so forth. The markets for these products are characterized as highly dynamic and with frequent changes in the needs of the customers. As a consequence, companies have to respond rapidly to changes in needs requiring them to be very flexible.

Due to this development, agile methods have emerged. In essence agile methods are light-weight in nature, work with short feedback and development cycles, and involve the customer tightly in the software development process. The main principles that

guided the development of different agile practices such as eXtreme programming [3] and SCRUM [28] are summarized in the agile manifesto [9]. As shown in a systematic review by Dyb and Dingsyr [8] agile has received much attention from the research community.

While agile became more and more popular lean software development has emerged with the publication of the book [24], which proposes ways of how practices from lean manufacturing could be applied in the software engineering context. Lean has a very strong focus on removing waste from the development process, i.e. everything that does not contribute to the customer value. Furthermore, according to lean the development process should only be looked at from an end-to-end perspective to avoid sub-optimization. The aim is to have similar success with lean in software development as was the case in manufacturing. That is, delivering what the customer really needs in a very short time.

Both development paradigms (agile and lean) seem similar in their goal of focusing on the customers and responding to their needs in a rapid manner. Though, it is not well understood what distinguishes both paradigms from each other. In order to make the best use of both paradigms it is important to understand differences and similarities for two main reasons:

- Research results from principles, practices, and processes shared by both paradigms are beneficial to understand the usefulness of both paradigms. This aids in generalizing and aggregating research results to determine the benefits and limitations of lean as well as agile at the same time.
- The understanding of the differences shows opportunities of how both paradigms can complement each other. For instance, if one principle of lean is not applied in agile it might be a valuable addition.

The comparison is based on the general descriptions of the paradigms. In particular, this chapter makes the following contributions:

- Aggregation of lean and agile principles and an explicit mapping of principles to practices.
- A comparison showing the overlap and differences between principles regarding different aspects of the paradigms.
- A linkage of the practices to the principles of each paradigm, as well as an investigation whether the practices are considered part of either lean or agile, or both of the paradigms.

The remainder of the chapter is structured as follows: Section 2 presents background on lean and agile software development. Section 3 compares the paradigms with respect to goals, principles, practices, and processes. Section 4 discusses the findings focusing on the implications on industry and academia. Section 5 concludes the chapter.

2.2 Background

Plan-driven software development is focused on heavy documentation and the sequential execution of software development activities. The best known plan-driven development model is the waterfall model introduced by Royce in the 1970s [27]. His intention was to provide some structure for software development activities. As markets became more dynamic companies needed to be able to react to changes quickly. However, the waterfall model was built upon the assumption that requirements are relatively stable. For example, the long lead-times in waterfall projects lead to a high amount of requirements being discarded as the requirements became obsolete due to changes in the needs of the customers. Another problem is the reduction of test coverage due to big-bang integration and late testing. Testing often has to be compromised as delays in earlier phases (e.g. implementation and design) lead to less time for testing in the end of the project.

In response to the issues related to plan-driven approaches agile software development emerged in the late 1990s and early 2000s. Agile software development is different from plan-driven development in many ways. For example, a plan-driven project contains detailed planning of the time-line with clearly defined products and documentation to be delivered while agile focuses on a high-level plan for the overall product development life-cycle with detailed plans only for the current iterations. Another difference is the way requirements are specified. That is, in plan-driven development there is a clearly defined specification phase where the complete requirements specification is created, the specification representing the contract. Hence, a change in requirements is a formal and work intensive process. On the other hand, agile welcomes changing requirements leading to continuous evolution. Consequently, change requests are handled through a more relaxed change request process. With regard to other activities (such as programming and testing) waterfall development concentrated these activities on one specific phase of development, while in agile development the activities are conducted throughout the overall development life-cycle [11]. The most prominent descriptions of agile software development process are eXtreme Programming (XP) [3] and SCRUM [28]. Each of these processes contains a high-level description of the work-flow and a set of agile software development practices. A mapping of

the practices and a description of the work-flows of different agile processes (including eXtreme Programming and SCRUM) can be found in [15] and [16].

Lean software development is inspired by ideas that have been used in the context of manufacturing and product development. Lean manufacturing led to tremendous performance improvement in the context of manufacturing cars at Toyota, the approach being referred to as the Toyota Production System. The lean manufacturing approach allowed delivering high quality products with fewer resources and in shorter time. The improvements were achieved by continuously improving processes through a systematic analysis focusing on waste identification; waste being everything that does not contribute to customer value. The ideas of lean manufacturing were put forward in the book The Machine that Changed the World (cf. [31]). In lean manufacturing the main focus was on optimizing the shop floor. Car manufacturers today have implemented the lean principles in their shop floors, i.e. the use of lean manufacturing does not lead to a competitive advantage anymore. Hence, to achieve further improvements the ideas behind lean have been extended to the overall product development life-cycle and the whole research & development organization. This includes the disciplines purchasing, sales and marketing, product planning, people management, and so forth. This view is more relevant for software development than the pure manufacturing view as in software engineering the overall development cycle should be in focus when improving software processes. The extensions to incorporate the whole of product development are known as the Toyota Product Development System (cf. [19]). The Poppendiecks translated the lean principles and practices known from manufacturing and product development to software engineering. Mary Poppendick stated that they were motivated to translate the practices when she heard about the waterfall model of software development, believing that software development would largely benefit from lean principles and practices, which helped product development in creating a flexible organization. The references from Mary and Tom Poppendieck (cf. [24, 25, 26]) are the main sources of how to interpret lean practices in the context of developing software. Hence, their books are used as the main sources for the identification of goals, principles, practices, and processes of lean software development. In comparison to agile, there are very few studies with an explicit focus on lean software development. The lack of empirical evidence for lean software development means that the comparison is focused on the generic descriptions provided in books.

2.3 Comparison

In this section the goals, principles, practices, and processes of the development paradigms are described and compared. The comparison focuses on different facets of the paradigms,

namely goals, principles, practices, and processes.

- Goals state what should be achieved by the paradigm. Hence, they present the rationale for why the principles, practices, and processes should be applied. In other words, goals represent the "why" (Subsection 2.1).
- Principles are rules that should be followed while using the paradigms. A rule of agile, for example, says that one should achieve technical excellence. Rules represent the "What" (Subsection 2.2).
- Practices are the implementation of the principles. For example, in order to implement technical excellence eXtreme programming uses pair programming to reduce the number of faults introduced into the code due to the continuous peer review process. Practices represent the "How" (Subsection 2.3).
- Processes describe the workflow and artifacts produced. That means the process is a representation of "when" an activity is done, and in which order (Subsection 2.4).

Each of the subsections of the comparison follows a similar pattern. First, the goals/principles/practices are introduced and thereafter the comparison is made.

2.3.1 Goals

Description of Goals

Goals describe why we should care about agile and lean; they provide a rational for software companies to adapt the paradigms. The following goals are identified for agile and lean:

- *Goal agile:* Agile (in comparison to traditional approaches such as plan-driven development) has the goal of delivering working software continuously that can be demonstrated to the customers to illustrate the latest status checking whether the software fulfills the customers' needs. Thus, the customers can provide feedback early and by that make sure that a product fulfills the needs of the customers. The goal becomes clear from the statement that working software is the primary measure of progress that should be delivered on regular bases in short cycles [9, 3, 15, 16].
- *Goal lean:* The goal of lean software development focuses on creating value for the customer rapidly and not spending time on activities that do not create value [24]. Value in this case has to be seen through the glasses of the customer [19].

> If there is an activity that is not of value for the customer then it is considered waste.

Comparison

Both goals have the focus on the customer in common. In the case of agile the customers are in focus as they should regularly receive working software. Lean adds the notion of value which was not as explicitly expressed in agile as it was in lean (see Table 2.1). Value has many different meanings and is a whole research field on its own, referred to as value-based software engineering [29]. One example of a definition of value in the lean context is that one should focus on everything that delights the customer, which is not necessarily what the customer wants or asks for [24, 19]. In the agile context, the needs of the customer are in the center, i.e. what the customer wants and requires. This is not in conflict with value, however, the notion of value puts more emphasis on exciting and delighting the customers and surprising them positively, which goes beyond satisfying the customers' needs.

Table 2.1: A Comparison of Goals for Lean and Agile

Aspect	Lean	Agile
Customer	Create value for the customer and thus only focus on value-adding activities.	Have a working product that fulfills the customers' needs.
Development speed	Rapid value creation and short cycle times	Continuous delivery of working software

Both development paradigms also share the goal of having frequent and rapid deliveries to the customer (see row Development Speed in Table 2.1). They are very similar in the sense that new and changed features should be made usable for the customer as fast as possible. This is a lesson learned from waterfall-oriented projects where all requirements are elicited, developed, tested, and finally delivering together based on a well defined plan. Waterfall development leads to a number of issues which were the reasons for the movement towards agile. The main issues are: (1) planned and validated requirements become obsolete as waterfall is inflexible in responding to changes; (2) reduction of test coverage due to limited and late testing; (3) the amount of faults increases with late testing; and (4) faults found late are hard and expensive to fix (see Chapter 3). In contrast, delivering fewer requirements/features more frequently avoids that requirements become obsolete, and allows for much earlier feedback from testing and customers [1].

The goals that we identified drive the principles by which lean and agile work. The principles of both development paradigms are analyzed in the following section.

2.3.2 Principles

Description of Principles

The principles constitute the rules that, according to the general descriptions of the methods, should be followed to achieve the goals (see Section 2.1). The principles for agile and lean are explicitly defined for both paradigms. The agile manifesto states four values and twelve related principles of agile development. In the lean context seven principles have been defined. For each principle we assigned a unique ID which eases the mapping and comparison between lean and agile, and also to make it easier to connect the principles to the practices implementing them. The following four values are presented in the agile manifesto [9]:

- *V1:* Individuals and interactions over processes and tools.
- *V2:* Working software over comprehensive documentation.
- *V3:* Customer collaboration over contract negotiation.
- *V4:* Responding to change over following a plan.

The agile manifesto states that the statements on the left have a higher value than the ones on the right. For example, agile does not say that there is no value in processes. However, as Koch [15] points out processes can be harmful if they are misused and hinder people in working together, drain people's enthusiasm and excitement, and require more investment in maintaining them than they help in working more effectively and efficiently. The 12 principles are based on the values and can be related to them. For each of the principles (AP01 to AP12) we state to which value the principle relates (cf. [9]).

- *AP01: Customer satisfaction:* The satisfaction of the customer should have the highest priority. To achieve this, software needs to be delivered early and continuously (i.e. software fulfilling the customer needs as stated in the goals). (V3)
- *AP02: Welcome change:* Changes should be welcomed by software developers, no matter if they come in early or late. The ability to react to late changes is seen as a competitive advantage. (V4)

- *AP03: Frequent deliveries:* Working software should be delivered frequently to the customer. Deliveries should happen within a couple of months or weeks. The manifesto stresses that preference should be given to the shorter time-scale. (V2)

- *AP04: Work together:* Developers (i.e. technicians) and business people (i.e. product managers, administration, management, etc.) are required to work together on a daily basis throughout projects. (V2)

- *AP05: Motivated individuals:* Motivated individuals are a prerequisite for successful projects and hence projects should be built around them. Building projects around them means to provide them with environments (e.g. tools and workspace) and to support them (e.g. project managers could help in avoiding unnecessary project disturbances). Furthermore, the project teams should be trusted to be successful in achieving the project goals. (V1)

- *AP06: Face-to-face conversation:* Face-to-face communication is seen as the most efficient way of exchanging information (e.g. in comparison to e-mail and telephone conversation). This is true within a development team, but also between teams and other relevant stakeholders of the project. (V1)

- *AP07: Working software:* The progress of software development should be measured through working software. Hence, working software is more important than detailed documentation, as was expressed in the second value of agile software development (V2).

- *AP08: Sustainable pace:* Everyone involved in the project (developers, software users, sponsors, managers, etc.) should be able to work indefinitely in a continuous pace. A process supporting the achievement of sustainable pace is referred to as sustainable development. (V1)

- *AP09: Technical excellence:* Agile requires discipline in focusing on technical excellence and good design. Having a high quality product and a good design allows for easy maintenance and change, making a project more agile. (V2)

- *AP10: Simplicity:* The agile manifesto defines simplicity as “the art of maximizing the amount of work not done - and is essential”. (V4)

- *AP11: Self-organizing teams:* Teams organizing themselves (e.g. picking their own tasks, and taking responsibility for completing the tasks) leads to the best requirements, architectures, and designs. (V1)

- *AP12: Continuous reflection:* Teams should reflect on their work continuously and think about how to become more efficient. Improvements should be implemented according to the discoveries made during the reflection. (V4)

Lean is based on seven principles, which are explained in more detail as they are not as self-contained as the ones presented for the agile manifesto.

- *LP01: Eliminate waste:* Waste in lean is everything that does not contribute to the value for the customer, i.e. everything that does not help to fulfill the needs of the customer or does delight the customer. Seven types of waste were identified in manufacturing and mapped to software development (see Table 2.2). The left column of the table describes the wastes in manufacturing and the right column the corresponding wastes in software engineering (cf. [24]). Each waste related to software development has an ID which is used to reference the wastes in the text. The wastes slow down the development flow and thus should be removed to speed up value creation.

- *LP02: Amplify learning:* Software development is a knowledge-intensive process where learning happens during the whole development lifecycle and needs to be amplified. Learning includes getting a better understanding of the customer needs, potential solutions for architecture, good testing strategies, and so forth. Thus, the processes and practices employed in a company should support learning.

- *LP03: Defer commitment:* A commitment should be delayed as far as possible for irreversible decisions. For example, a tough architectural decision might require some experimentation and therefore should not be committed early. Instead, the option for change should be open for as long as possible. [24] point out that not all decisions are irreversible and thus do not have to be made late as they can be changed.

- *LP04: Deliver as fast as possible:* Lean has a strong focus on short cycle times, i.e. to minimize the time from receiving a request for a feature to the delivery of the feature. The reason for the strong focus on cycle time is that while a feature is under development it does not create value for the customer.

- *LP05: Respect people:* Poppendieck and Poppendieck [24] provide three principles that were used in the context of the Toyota Product Development System fostering the respect for people: (1) Entrepreneurial leadership: People that are

led by managers who trust and respect them are more likely to become good leaders themselves. This helps in creating a management culture facilitating committed and independent people in an organization. (2) Expert technical workforce: Successful companies help building expertise and managers in these companies make sure that the necessary expertise for achieving a task is within the teams. (3) Responsibility-based planning and control: Management should trust their teams and not tell them how to get the job done. Furthermore, it is important to provide the teams with reasonable and realistic goals.

- *LP06: Build quality in:* Quality of the software product should be built in as early as possible, and not late in development by fixing the defects that testing discovered. In result the integrity of the software in development should be high at any point in time during the development lifecycle. As [24] point out, a prerequisite for achieving integrity is very high discipline. For example, if a defect is discovered early in the development process the ongoing work must be stopped and the defect fixed.

- *LP07: See the whole:* When improving the process of software development the whole value-stream needs to be considered end to end (E2E). For example, there is no point in sub-optimizing the requirements process and by that increase the speed of the requirements flow into coding and testing if coding can only implement the requirements in a much slower pace.

Comparison

Figure 2.1 shows a mapping of the principles related to lean and agile. The identified principles were grouped into seven aspects (people management and leadership; quality of the product; release of the product; flexibility; priority of the customer needs/value; learning; and E2E flow). Each aspect contains a set of principles for lean and agile. If principles from the lean and agile paradigms respectively are stated in the same row then they are related and their relationship is explained further. For example, AP11 (self-organizing teams) and LP05 (respect people) are related within the aspect "people management and leadership". We can also see that there exists an N to N relationship between the principles of both paradigms, e.g. AP07 (working software) can be related to LP01 (eliminate waste) and LP06 (build quality in). Vice versa the principle LP01 (eliminate waste) relates to several agile principles (e.g. AP03 - frequent deliveries and AP01 - customer satisfaction) . If only one column states a principle then the principle is only explicitly referred to in one of the development paradigms, such as LP07 (see the whole). For LP01 (eliminate waste) we also state which waste is concerned in the

Table 2.2: Wastes in Lean Software Engineering and their Mapping to Manufacturing (cf. [24]

Manufacturing	Software Engineering
Inventory: Intermediate work-products and work in process	*W1: Partially Done Work:* Work-in-process that does not have value until it is completed (e.g. code written, but not tested)
Over-Production: The number of produced items is higher than the number of demanded items (inventory in this case is "dead capital"	*W2: Extra Features:* Functionality that has been developed, but does not provide value to the customer
Extra Processing: Extra work is created in the production due to e.g. poor set-up of machines	*W3: Extra processes:* Process steps (e.g. creation of documentation that is not really needed) that can be removed
Transportation: Transport of intermediate work-products (e.g. due to a poor layout of the production line)	*W4: Handovers:* Many handovers (e.g. documentation) create overhead
Motion: People and machines are moved around instead of being used to create value	*W5: Motion/Task Switching:* People have to move to identify knowledge (e.g. team members that work together are not co-located) or have many disturbances in their work
Waiting: A machine with free capacity is waiting for input	*W6: Delays:* There are delays in development that, for example, cause waiting times within a development team (team idles)
Defects: Fixing of problems in the products	*W7: Defects:* Fixing of problems in the products

comparison. In the following paragraphs we explain why and how the principles in Figure 2.1 are related to each other. The IDs in Figure 2.1 (AP01 to AP12 and LP01 to LP07) refer to the detailed descriptions of the principles provided earlier.

People management and leadership: This aspect contains all principles that are related to leading and managing people in a project. As can be seen in Figure 2.1 for each of the principles of agile a corresponding principle of lean development can be identified.

- *Relation of AP05 (motivated individuals), AP08 (sustainable pace), and AP11 (self-organizing teams) to LP05 (respect people):* Respecting people (LP05) is facilitated by trusting the team to find a solution for a given task (Responsibility-

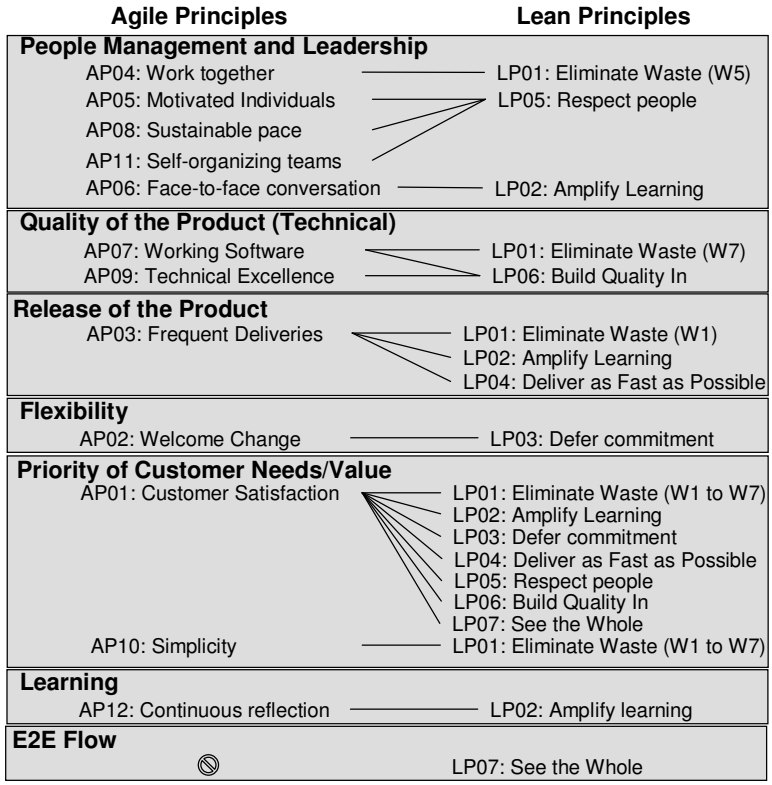


Figure 2.1: Mapping of Agile and Lean Principles

Based Plan and Control). This is the same as self-organizing teams (AP11) who take on the responsibility for solving a task in agile development. Respecting people (LP05) is also connected to sustainable pace (AP08) as self-organized teams that are trusted and empowered are more motivated over a long period of time (AP05 - motivated individuals) and thus it can be expected that they are working productively in a continuous manner.

- *Relation of AP06 (face-to-face conversation) to LP02 (amplify learning):* Face-to-face conversation (AP06) allows for direct and instant communication to resolve misunderstandings and thus amplifies learning (LP02). For example, informal communication taking place in coffee corners is considered an important

part of information exchange allowing people to share knowledge [7].

- *Relation of AP04 (work together) to LP01 (eliminate waste "motion/task switching"):* Agile raises the importance of different groups of people (e.g. technical developers and business/marketing) to work closely together (AP04). Close cooperation between people of different competence areas (LP01 - eliminate waste) helps in making their competence more easily accessible which reduces the time to find the information. Consider the example of a developer who should decide which requirement to implement next based on its importance to the market. Without cooperation the developer would have to spend time searching for documentation and/or the right person to ask. Having cooperation between marketing and development in the first place would allow for easy and quick access to the information by just asking the marketing-representative in the team.

Comparison: Every principle in lean has a corresponding principle in agile for the aspect "People Management and Leadership". In conclusion both paradigms very much share the same rules when it comes to managing people.

Technical quality of the product: This aspect contains the principles that are related to achieve a working product with high quality from a technical perspective. Figure 2.1 shows that lean and agile both apply principles related to technical product quality.

- *Relation of AP09 (technical excellence) to LP06 (build quality in):* Agile stresses that technical excellence should receive continuous attention (AP09). In lean this is achieved by building in quality early in the development process, and not by testing for and fixing defects later (LP06). Thus, LP06 fulfills the principle of having a continuous focus on building technically excellent products.

- *Relation of AP07 (working software) to LP01 (eliminate waste) and LP06 (build quality in):* The rule of having working software (AP07) throughout development enforces that the quality of the software has to be ensured throughout the whole development lifecycle. For example, having mechanisms in place to increase the quality of code while it is written (LP01 - eliminate waste) helps to achieve the goal of working product with few defects (LP06, W7).

Comparison: Both paradigms stress the continuous attention to quality and technical excellence. A consequence of this attention is a working software product throughout the development lifecycle. Thus, both paradigms strongly agree on the rules applied to the quality of the software product. A small distinction is made in principle AP09 (technical excellence) where agile emphasizes that good design enhances agility. For example, an easy extension of the architecture enables a rapid and agile response to changing customer needs.

Release of the product: The release aspect refers to the delivery of software to the customer. The release aspect is also covered in both development paradigms.

- *Relation of AP03 (frequent deliveries) to LP01 (eliminate waste), LP02 (amplify learning), and LP04 (deliver as fast as possible):* Frequent deliveries to the customer (AP03) have a positive effect on the elimination of waste concerning partially done work (LP01). That is, enforcing frequent deliveries avoids that completed work (e.g. adding a new feature) stays within the development organization without being made available to the customer. Frequent deliveries also amplify learning (LP02) as they allow the customer to provide regular feedback on the latest status of the product. Thereby, the development organization can learn about the needs of the customers and what features excite them. In addition AP03 influences the speed of delivery (LP04) positively. In waterfall development the delivery of the overall scope is done in the end which means that all features together have a very long lead-time. Frequent deliveries, however, imply that less software is delivered at once, but much more frequently and with shorter lead-time (see Chapter 5).

Comparison: There is a clear relation between the principles of the release aspect between lean and agile, i.e. both paradigms are in strong agreement on this.

Flexibility: Flexibility is the ability to react on changes that impact the development of the software product. The most common is the change in the needs of the customer reflected in changing requirements. Other changes are time line changes (e.g. deadlines), changes of rules and regulations (law) that affect development, or innovations in technology.

- *Relation of AP02 (welcome change) to LP03 (defer commitment):* Agile stresses that software organizations should welcome change instead of fighting it (AP02) as being able to deliver what the market needs today determines the success of products. The principle refers to the attitude that one should have when developing software. Lean adds to that by providing a rule that supports the attitude of welcoming change, i.e. to defer commitment (LP03). Deferring commitment means to decide as late as possible. For example, a company should not decide of an overall release scope early on in development (early decision), but instead decide whether a feature should be included into the scope as late as possible (deferred commitment).

Comparison: Both paradigms address flexibility, but describe it in a different way. In lean one should defer commitment, i.e. decide as late as possible. However, software organizations that are generally driven by plans, scopes, and deadlines have to change

their attitude towards harnessing change as this is a prerequisite to implement late decisions. Hence, the principles of the two development paradigms complement each other very well as accepting and harnessing change in the process is a pre-requisite to defer commitment.

Priority of customer needs/value: This aspect contains principles that stress the priority of the customer in software development over other focuses (such as the focus on documentation).

- *Relation of AP01 (customer satisfaction) to LP01 (eliminate waste, all wastes in Table 2.2):* The priority of the customer (AP01) is reflected in all the principles that are stated for lean (LP01-LP07). This is very clear for the wastes (LP01) as the waste is identified from the point of view of the customer. Amplify learning (LP02) puts high priority on customers' needs as it is about learning the needs of customers and what delights them. The same holds for deferred commitment (LP03) as this enables a flexible reaction to customer change requests that are due to a change in needs. Delivering fast (LP04) implies that the needs of the customer are realized quickly as soon as they are articulated. The respect for people (LP05) can also be related to the customer focus as happy employees are an important factor for project success [7]. Furthermore, technical quality (LP06) is a prerequisite to deliver valuable software (e.g. if the software is not stable then the customer cannot make sufficient use of its functionality). Finally, see the whole (LP07) implies that one should not put too much effort in sub-optimization of the process as this is does not lead to significant improvements for the customer. Thus, the improvement effort would be wasted.

- *Relation of AP10 (simplicity) to LP01 (eliminate waste, all wastes in Table 2.2):* Simplicity is about maximizing the amount of work not done (AP10). In that sense it is very strongly related to wastes (LP01) as the elimination of waste leads to a reduction of work. This includes unnecessary work that can be easily avoided (e.g. documentation never used) or reduction of rework (e.g. defects).

Comparison: The comparison indicates that customer priority in agile is related to all principles in lean. As we have shown the principles in lean are also linked to the principles in agile other than AP10 we conclude that AP10 is the very central principle of both development paradigms.

Learning: This aspect is about gaining new knowledge (e.g. about customer needs, ways of working, etc.) and is addressed in both paradigms.

- *Relation of AP12 (continuous reflection) to LP02 (amplify learning):* Continuous reflection allows time for the team to reflect on how to improve the ways

of working to become more efficient (AP12). Thus, the learning focus is on identifying improvement potential for efficiency. In lean the learning focus has a broader perspective by emphasizing learning in general, not with a specific focus on efficiency. Learning in general includes, for example, gaining new knowledge about customer needs and market trends.

Comparison: Both paradigms focus on learning, while lean takes a more general perspective.

E2E flow: The E2E flow includes principles that emphasize the focus on the overall flow of value (i.e. from the very beginning when a need for a feature enters the organization till it is delivered). The principle related to the E2E flow is "see the whole" (LP07). When comparing the principles within the aspect "priority of customer needs/value" LP07 is related to the prioritization of the customer needs (AP01 - customer satisfaction). However, the E2E flow aspect is not considered in the principles of agile and thus is what sets lean and agile apart when looking at the principles.

Overall, the comparison shows which principles are the same, complement each other, or are new to either one of the two paradigms.

- *Same:* For the aspects people management and leadership, technical quality of the product, and release of the product both paradigms strongly agree on the principles. That is, they mean the same, but only express it in different words.

- *Complementary:* The paradigms complement each other with regard to the aspects flexibility, priority of customer needs/value, and learning. For flexibility lean emphasizes deferred commitment, while agile stresses the attitude a company must have to be willing to defer commitments. For priority of customer needs/value lean complements agile by concretizing what does not contribute positively to the value for the customer in the form of the seven wastes of software development (see Table 2.2).

- *New:* The need to look at the development and value flow from an end to end perspective is unique for lean and therefore is what clearly distinguishes both paradigms from each other.

Regarding the question whether lean development is agile and agile development is lean we can provide the following answer for the principles: Lean is agile as it includes all the principles of agile. However, agile is not lean as it does not emphasize the E2E focus on flow in its principles.

In the next section we analyze the practices which implement the principles of lean and agile development. A proposition based on the similarities is that both paradigms also propose similar practices. However, this proposition has to be investigated as:

- Agile and lean might propose different practices ("How") in order to fulfill the practices they agree on ("What").
- Lean is unique in its E2E focus and hence we can expect to identify practices that are not already proposed in the agile context.

Furthermore, it is interesting to investigate which principles are covered by which practices, as this investigation shows the coverage of practices through principles.

2.3.3 Practices

Practices describe how the principles are implemented by the development paradigms. Therefore, we first present the practices for lean and agile and link each of the different practices to the principles. After that we make a comparison between the paradigms.

In total we identified 26 principles by looking at the literature describing lean [24, 25, 26] and agile software development [3, 28, 16, 15], as well as lean product development [19]. The principles are described and for each principle it is stated whether literature connects it to lean, agile, or both paradigms. The principles are grouped as being related to requirements engineering, design and implementation, quality assurance, software releases, project planning, team management, and E2E flow. First, a comparison of practices in each group is made, and thereafter we provide an overall comparison.

Requirements Engineering

P01: On-site customer: Representatives of the customer are located at the development site to allow for immediate feedback on the product (AP02 - welcome change). At the same time the customer always knows about the progress of the development (AP01 - customer satisfaction). The co-location also allows the developers to interact with the customer to ask questions and clarify requirements, which avoids implementation of features not needed by the customer (LP01 - eliminate waste "extra features"). Furthermore, a regular face-to-face communication between the team and the customer is ensured (AP05 - face-to-face conversation).

P02: Metaphors and user stories: A metaphor is a very high level requirement outlining the purpose of the system and characterizes what the system should be like. The purpose on the high level should be stable. The metaphor is broken down into more detailed requirements to be used in the development project. These are feature descriptions (FDD) or user stories (XP and SCRUM). The features/user stories should be used to track the progress and apply the pull concept (see P26 - Kanban pull-system). Having the metaphor defined also avoids the inclusion of irrelevant user stories (LP01

- eliminate waste "extra features") and describes what should be developed to provide value to the customer (AP01 - customer satisfaction). That is, if a user story cannot be linked to the metaphor then it should not be included in the product.

Comparison: Table 2.3 shows the principles related to the requirements practices, and whether the principles are considered in lean and agile development. Metaphors and user stories have been recognized in agile as well as lean software development. However, the on-site customer is not part of the lean practices, but is a key practice in agile software development. Both practices support lean and agile principles.

Table 2.3: Comparison for Requirements Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in agile SE	Used in lean SE
P01: On-site customer	√					√							√	√						√	
P02: Metaphors/Stories	√												√							√	√

Design and Implementation

P03: Refactoring: Refactoring is the continuous improvement of already working code with respect to maintainability, readability, and simplification of code (AP10 - simplicity) which has a positive effect on understanding the code (LP02 - amplify learning). When code is being worked on over a long period of time the assumptions that were made in the past while writing the code might not be true in the end. For example, a class that was written in the beginning of a project might have to be restructured in order to fit the latest version of the overall product in a better way, e.g. removing duplicated code, changing code to improve readability, or changing the structure of the class to fit a certain design pattern (cf. [1]). It is important to mention that changing the external structure (e.g. interfaces and their parameters) should be avoided as this might be harmful for the integrity of the overall system. The simple and clear structure of the code helps new developers to become more productive in delivering value (AP01 - customer satisfaction).

P04: Coding standards: Coding standards make sure that developers structure and write code in the same way. This is important to assure that everyone can understand

the code (AP01 - customer satisfaction, LP02 - amplify learning). Furthermore, a common understanding on how to code avoids unnecessary discussions in pair programming. Examples for coding standards for Java are [18]:

- File organization (e.g. package statements before import statements)
- Interface declarations (public before protected before private variable declarations)
- Wrapping lines
- Rules for formatting if-else, try-catch, loop-statements, etc.
- Naming conventions (packages, classes, interfaces, methods, etc.)

Understandability and maintainability can be improved further by applying good programming practices in addition to the rules for formatting and naming. For the Java Code Convention different rules apply for programming practices (e.g. one should avoid to use an assignment operator in a place where it can be confused with an equality operator; one should not assign a value to several variables at once; always use parentheses with mixed operators; only use return once in a method; etc.).

P05: Team code-ownership: The code written by an individual is not owned by that individual. Instead, everyone in the team owns the code and is allowed to make changes. Team code ownership is also related to the concept of egoless programming [30] where the success as a team is more important than promoting the status of the team member with the strongest ego. The team factor is a driver for motivation (AP01-customer satisfaction, AP05 - motivated individuals) and gives credit to each member of the team for the achieved result (LP05 - respect people).

P06: Low dependency architecture: This type of architectures clearly encapsulates functionality into components that can be developed independently, i.e. the delivery of one component does not depend on the delivery of another component [26]. That way the functionality provided by the components can be delivered to the customer as soon as they are ready (AP01 - customer satisfaction, AP03 - frequent deliveries) and by that reduce the amount of partially done work (LP01 - eliminate waste). Furthermore, the architecture becomes easier to change (AP03 - frequent deliveries) as the change impact is more isolated with few dependencies.

Comparison: Table 2.4 shows the principles linked to the design and implementation practices, and whether the principles are considered in lean and agile development. Refactoring has been considered in lean as well as agile (cf. [24]). Coding standards and team-code ownership are unique to agile software development [15], while low dependency architecture is a principle unique to lean [26]. All practices identified for design and implementation are linked to lean as well as agile principles.

Table 2.4: Comparison for Design and Implementation Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in lean SE	Used in agile SE
P03: Refactoring	√									√				√						√	√
P04: Coding standards	√	√															√				√
P05: Team-Code Own.	√				√																√
P06: Low Dep. Arch.	√	√	√							√			√			√				√	

Quality Assurance

P07: Test-driven development and test automation: In test-driven development (TDD) [4] the test cases for unit tests are written before the implementation takes place. The test cases are to be implemented (e.g. JUnit [5] is a test framework supporting the implementation in Java) so that they can verify the implementation as soon as it is finished (AP07 - working software, AP09 - technical excellence, L06). Thereby, defects are caught early (LP01 - eliminate waste "delays"), which results in higher quality software for the customer (AP01 - customer satisfaction). Whenever the implementation is changed the test cases can be re-run as they are already implemented. This aids in automation of regression tests. After the test is completed the code is refactored (see P03 - refactoring).

P08: Pair-programming: In pair programming two developers share one workstation. One of the developers is actively developing the test cases and writing the code. The second developer should reflect on what the first developer is doing and act as a reviewer thinking about the impact (e.g. how does the implementation affect other parts of the system) and the quality (e.g. is the code defective, or are any important unit test cases missing). The review allows to detect defects in the code immediately after their introduction (AP09 - technical excellence, LP01 - eliminate waste "defects", LP06 - build quality in) improving the quality for the customer early on in the coding process (AP01 - customer satisfaction). The second developer can improve the work by asking questions and providing feedback on the work done. Agile recommends to continuously changing pairs throughout a project to improve knowledge transfer (LP02 - amplify learning).

P09: Continuous integration: Features that are developed as increments for a product should be integrated into the overall product as soon as possible after they are finalized, making every extension to the product deliverable (AP01 - customer satisfaction, AP03 - frequent deliveries, LP01 - eliminate waste "partially done work"). That way, problems in integration are discovered early and can be fixed close to their discovery (AP07 - working software, AP09 - technical excellence, LP01 - eliminate waste "partially done work", and LP06 - build quality in). Furthermore, integrating a large scope at once often leads to unpredictable results in terms of quality and schedule [23].

P10: Reviews and inspections: Inspections are a visual examination of any software artifact (requirements, code, test cases, etc.) allowing the detection of defects early in the process (AP01 - customer satisfaction, AP07 - working software, AP09 - technical excellence, LP01 - eliminate waste "defects", and LP06 - build quality in). Fagan introduced a very formal inspection process in 1976 [10]. However, the formality requirements are not given in the agile context, i.e. different agile methods use reviews and inspections in a different way. The inspection in Feature Driven Development (FDD) [15] is a relatively rigorous peer review. Furthermore, post-mortems are used in FDD where completed development activities are reviewed to identify improvement potential for upcoming iterations. SCRUM also uses reviews after each 30 day sprint where the work product of the sprint is reviewed by all relevant stakeholders during a meeting [28].

P11: Configuration management: The goal of configuration management is to achieve consistency between versions of software artifacts and thus to achieve system integrity [17]. Different software artifacts linked to a configuration item should be consistent in the sense that requirements, code, and test cases match and represent the same version of the system. For example, an inconsistent situation would be if test cases are derived from a specific requirements specification, but are linked to an older version of the requirements specification within the configuration. The negative consequence is that the wrong things would be tested. In order to achieve and maintain consistency configuration management has mechanism for version control and change processes. Configuration management is not explicitly acknowledged in most of agile process models, only FDD has configuration management as one of its practices [15]. The benefits of configuration management are: (1) create transparency in terms of status and progress of configuration items to be reported to the customer (AP01 - customer satisfaction); (2) the knowledge about status avoids making the wrong decisions causing waiting (LP01 - eliminate waste "delays"), quality problems (AP07 - working software, AP09 - technical excellence, LP01 - eliminate waste "defects", and LP06 - build quality in); and (3) the information of configuration items aids in communication and reflection (AP12 - continuous reflection).

Comparison: Table 2.5 shows the principles linked to the quality assurance prac-

tices, and whether the principles are considered in lean and agile development. All principles have been considered in lean as well as agile (cf. [15, 16, 24]). Furthermore, all quality assurance practices are linked to lean as well as agile principles.

Table 2.5: Comparison for Quality Assurance Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in lean SE	Used in agile SE
P07: TDD	√						√		√				√					√		√	√
P08: Pair-programming	√								√				√	√				√		√	√
P09: Continuous Int.	√		√				√		√				√		√					√	√
P10: Reviews/Insp.	√						√		√				√					√		√	√
P11: CM	√						√		√				√					√		√	√

Software Releases

P12: Incremental deliveries to the customer: New functionality is delivered continuously as an increment of the previous software version to the customer. The increments go through a development lifecycle consisting of the activities requirements elicitation, design and integration, implementation, testing, and release. The flow of activities (e.g. order of activities, branching, merging, and loop-backs) depends on the process model used. Frequent deliveries help to achieve customer satisfaction as the customer receives value continuously (AP01 - customer satisfaction, LP01 - eliminate waste "partially done work") and speed up deliveries of value (AP03 - frequent deliveries, LP04 - deliver as fast as possible). Furthermore, incremental deliveries assure that a working system is build continuously with each new increment (AP07 - working software, LP06 - build quality in). Learning is amplified as increments allow the customer to provide feedback on the integration of each new feature (LP02 - amplify learning). As features are developed rapidly after requesting them the risk of features not needed is reduced (LP01 - eliminate waste "extra features").

P13: Separation between internal and external releases: Internal releases are baselines of the software product that have the quality to be released to the market, but are not. One reason to not releasing them could be that the market window is not right from

a marketing strategy. The internal release makes sure that there is a continuous attention to quality as baselines should be releasable (AP01 - customer satisfaction, AP09 - technical excellence, LP01 - eliminate waste "defects", and LP06 - build quality in). An example of the implementation of internal and external releases in a market-driven context can be found in [21].

Comparison: Table 2.6 shows the comparison of software release practices. The principles can be found in both paradigms (cf. [15, 24, 16]). Furthermore, the practices fulfill principles of lean and agile.

Table 2.6: Comparison for Software Release Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in lean SE	Used in agile SE
P12: Inc. Del.	√		√				√						√	√		√		√		√	√
P13: Int./Ext. Rel.	√								√				√					√		√	√

Project Planning

P14: Short iterations: Within an iteration the product is further improved and enhanced based on regular feedback and planning of the iteration (see e.g. sprints in SCRUM [28]). For example, a feature is delivered to the customer to receive feedback, the feedback being that the feature needs to be improved. Based on the feedback from the customer the feature goes into another iteration to incorporate the feedback (AP02 - welcome change). When the customer is satisfied (AP01 - customer satisfaction, AP07 - working software) no further iteration is required. The iteration cycles should be short (no more than a month with a preference for a shorter scale) to allow continuous feedback (AP02 - welcome change) to improve the product (AP09 - technical excellence, LP06 - build quality in, LP01 - eliminate waste "defects"). The feedback allows the team to learn about the needs of the customer (LP02 - amplify learning). The shortage of the iteration assures that work is completed and made available to the customer in a rapid manner (LP01 - eliminate waste "partially done work").

P15: Adaptive planning with highest priority user stories / requirements: A list of requirements is maintained which shows the priority of the requirements, the require-

ments that are more important for the customer being higher ranked in the list (AP01 - customer satisfaction). The priority of the requirements is important to indicate which requirement should be implemented next. The list is the backlog of work. Adaptive planning means that the priority of requirements can be changed in the backlog and there is flexibility in choosing work-tasks for the next iteration (AP02 - welcome change, LP03 - defer commitment). Furthermore, adoption avoids delivering features the customer does not need (LP01 - eliminate waste "extra features") while learning more about the customers' needs (LP02 - amplify learning).

P16: Time-boxing: Fixed start and end dates are set for iterations and projects. SCRUM, for example, proposes to have 30 day sprints in which the next increment has to be completed [28]. In consequence the scope of development (i.e. how much to implement in a project) has to be decided based on the maximum duration of the project. Hence, time-boxing forces new functionality to be delivered within one cycle speeding up value creation (AP01 - customer satisfaction) and the rate in which new functions can be delivered (AP03 - frequent deliveries,LP01 - eliminate waste "partially done work").

P17: The planning game: In the planning game the next iteration of the project is planned. Different stakeholders have to be involved in the planning game, namely the customer, developers, and managers. The game is usually organized in a workshop setting and allows the participants, that are not on-site during the whole development time, to meet all important stakeholders (AP05 - face-to-face conversation, LP01 - eliminate waste "extra processes") [15]. Furthermore, the meeting is used to resolve conflicts and assures that the right feature is developed in the next iteration (LP01 - eliminate waste "extra features"). It can also be used to reflect on the previous iteration as the baseline for the next one (AP12 - continuous reflection) and by that supports learning (LP02 - amplify learning). A regular planning game meeting allows the customer to suggest changes to the plan (AP02 - welcome change).

Comparison: Table 2.7 shows the comparison of project planning practices. The practices short iterations, adaptive planning, and time-boxing can be found in both paradigms (cf. [15, 16, 24]). The planning game is unique to agile, the practice being used in SCRUM [28]. Furthermore, the practices fulfill principles of lean and agile.

Team Management

P18: Co-located development: People from different disciplines should be located together to ease communication. Direct communication can replace documentation that otherwise would have to written and handed over between disciplines, which reduces the number of handovers and extra processes to create the documentation (LP01 - eliminate wastes "extra processes" and "handovers") (see Chapter 5). An additional benefit

Table 2.7: Comparison for Project Planning Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in lean SE	Used in agile SE
P14: Short iterations	√	√	√				√		√				√	√				√		√	√
P15: Adaptive Planning	√	√											√	√	√					√	√
P16: Time-boxing	√		√										√							√	√
P17: Planning game	√	√				√						√	√	√							√

is that people do not have to move around or wait to discuss (motion/task-switching), which eases learning from each other (LP02 - amplify learning). With the removal of this waste one can concentrate on value-adding activities instead (AP01 - customer satisfaction). The continuous discussions between team members also aid in the continuous reflection with regard to the ways of working (AP12 - continuous reflection).

P19: Cross-functional teams: Teams need to be cross-functional so that they gather the competence needed to develop a feature. Two situations are shown in Figure 2.2. In the first situation (top of the figure) a development-centric organization is shown where other disciplines, such as requirements and testing, exchange information with the development team. There is also information exchange between each of the disciplines, which is not included in the figure for simplification. The separation of the development team (coding) from other disciplines hinders the exchange of information (e.g. in the form of barriers and lack of understanding for each other) (cf. [8, 21]). Consequently, it is beneficial to create cross-functional teams where each discipline has at least one representative in the team (see bottom of the figure) [19, 13]. The representation of different disciplines helps them in learning from each other (LP02 - amplify learning), and assures that business and technical people work together and develop an understanding for each other's work (AP04 - work together). Furthermore, people do not have to search long to find the right competence (LP01 - eliminate waste "motion/task switching") and do not have to rely on handovers and documentation (LP01 - eliminate wastes "extra processes" and "handovers"). Overall, the avoidance of handovers and extensive documentation frees resources for value-adding activities (AP01 - customer satisfaction).

P20: 40-hour week: Overtime should be avoided as rested people produce better

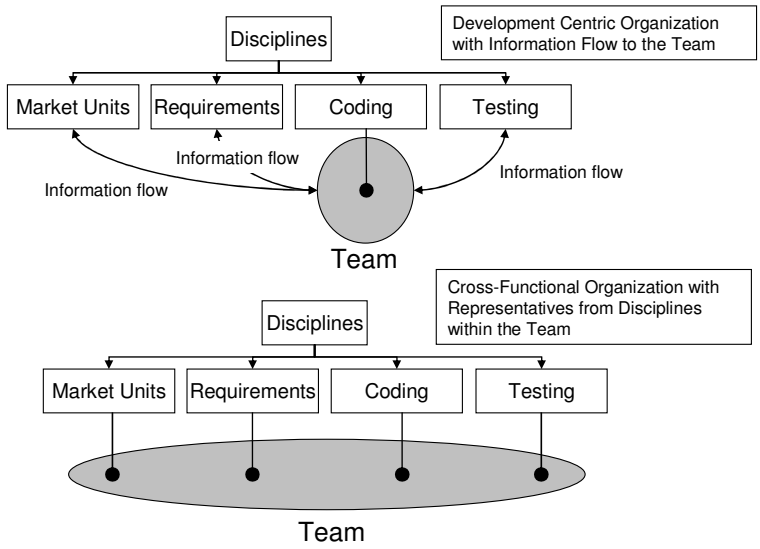


Figure 2.2: Cross-Functional Teams

work and are more concentrated, reducing the number of mistakes made (AP09 - technical excellence, LP01 - eliminate waste "defects", LP06 - build quality in),. Thus, the working time in the projects should be limited to approximately 40 hours. If developers do overtime in one week they are not allowed to do overtime in the week after that. Koch [15] distinguishes between forced and voluntary overtime and states that voluntary overtime should be tolerated. Rested developers can produce quality over a long period of time, while overworked developers might end up in a burnout situation. The avoidance of burnouts aids in achieving a sustainable pace (AP08). Furthermore, ongoing deliveries to the customer are assured (AP01 - customer satisfaction).

P21: Standup-meeting: This meeting takes place daily and is a place for the whole team to communicate and reflect on the completed and ongoing work (AP12 - continuous reflection). The discussions in the meeting should aid in becoming more efficient. The meeting should not last longer than 15 minutes. Participants are standing during the meeting to ensure that the meeting is kept short. Every team member has to answer three questions:

- What has been achieved since the previous meeting?
- What will be achieved before the next meeting?

- What were hinders in achieving your goals?

The last question should help the manager in supporting the team member. For example, one hinder could be the disturbance of a project by additional tasks. The manager then has to make sure that the project is not further disturbed. Furthermore, wastes like waiting times could be raised. Hence, stand-up meetings aid in the discovery of the different wastes not generating value (AP01 - customer satisfaction, LP01 - eliminate waste).

P22: Team chooses own tasks: Teams should be able to choose their own tasks, which increases the commitment to the task (AP05 - motivated individuals, LP05 - respect people). For example, from a number of features to be developed a team can choose which feature to implement, given some restrictions regarding the priority of the features. The choice is also influenced by the competencies in the team so that the task is implemented in the most efficient way for the customer (AP01 - customer satisfaction). The team takes the responsibility to complete the task (A11 - self-organizing teams).

Comparison: Table 2.8 shows the comparison of project planning practices. Three practices are shared between lean and agile, namely co-located development, cross-functional teams, and team chooses own tasks (cf. [15, 16, 24]). The 40-hour week [3] and stand-up meetings [28] are unique to agile. The practices fulfill principles of lean and agile.

Table 2.8: Comparison for Team Management Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in lean SE	Used in agile SE
P18: Co-loc. develop.	√					√						√	√	√						√	√
P19: Cross-func. teams	√			√									√	√						√	√
P20: 40-hour week	√							√	√				√					√			√
P21: Stand-up meeting	√											√	√								√
P22: Team chooses T.	√	√			√						√						√			√	√

E2E Flow

P23: Value-stream mapping: A value-stream map visualizes the end-to-end flow of the overall development lifecycle [19], i.e. the emphasis is on seeing the whole (LP07). The map shows the activities and lead-times. The lead-times are separated into processing time and waiting time. Processing time means that work is done during that time (e.g. a requirement is implemented). Waiting time means that nobody works on the requirement (e.g. the requirement has been implemented and is waiting to be tested). The processing time value is added to the product while waiting time is non-value adding. The overall lead time is the sum of all waiting and processing times. The map should help in identifying everything that does not contribute to the value of the customer (AP01 - customer satisfaction). A value-stream map analysis is conducted in four steps:

- *Create current-state value-stream map:* First, the value-stream map is created by following the steps a single work product (e.g. requirements, test cases, change requests, etc.). The data is collected through interviews across the lifecycle of the work product. If lead time measurements are available then these should be considered as well.

- *Analyze current-state value-stream map:* The map is analyzed by looking at critical wastes as those have the largest improvement potential. The criticality of lead times from an end to end perspective can be determined by calculating the ratio of non-value adding time and overall lead time.

- *Identify reasons for waiting times and propose improvements:* The reasons for waiting times and long processing times are identified in workshops and discussions with practitioners, e.g. in form of a root cause analysis [1]. During the root cause analysis all types of waste can be discovered by having a dialogue about the map. For example, by asking an engineer why a certain activity takes so long the engineer can pinpoint many types of waste (LP01 - eliminate wastes W1 to W7). Depending on the nature of the problems for long processing and waiting times the value stream map can potentially pinpoint to reasons that, when being addressed, fulfill any of the principles of agile and lean development (AP01 to AP12 and LP01 to LP07)

- *Create future-state map:* Solutions are proposed for the identified problems and the potential of the solution candidates in reducing the waiting and processing times is evaluated. Based on the assessment of the improvement suggestions a new map (future-state map) can be constructed. The future-state map provides

an indication of the overall improvement potential in terms of lead-time when implementing the solutions.

An example of an analysis of a software process for product customizations with value-stream maps and the notation to illustrate them can be found in [20].

P24: Inventory management with queuing theory and theory of constraints: Inventory is the work in process that is not completed and thus does not provide value to customers. High inventory levels have a negative impact on the performance of software development for multiple reasons: (1) Inventory hides defects (hinders fulfilling AP07 - working software, AP09 - technical excellence, LP01 - eliminate waste "defects", and LP06 - build quality in); (2) Time and effort has been spent on the artifacts in the inventory and when not making them available to the market they might become obsolete (hinders fulfilling AP03 - frequent deliveries, LP01 - eliminate waste "partially done work", and LP04 - deliver as fast as possible) [23]; (3) Inventory creates waiting and slows down the development flow (hinders fulfilling AP03 - frequent deliveries, LP01 - eliminate waste "delays", and LP04 - deliver as fast as possible). Point (3) can be related to queuing theory where one can calculate the work-in-process and the time work stays in process based on arrival distributions and processing times (distribution of time needed for the server, e.g. a development team, to complete a task), and by that identify overload situations. An example of identifying overload situations with queuing theory based on real world data for a requirements process is shown in [12]. In order to achieve a continuous flow of development the workload should be below the capacity of the development organization (see Chapter 7). Being below the capacity also allows for emergency situations (e.g. customer faults, emerging high priority requirements).

The theory of constraints (TOC) also helps to reduce inventory levels. TOC consists of five steps: (1) Identify the system constraint; (2) Decide candidate solutions of how to remove the constraint; (3) Select candidate solution that is focused on constraint; (4) Remove constraint with selected solution; (5) go to step (1) to see whether a new system constraint has become visible [2]. The constraint can be identified by looking at processing times of different steps in the development flow. For example, how many requirements are processed per time unit (e.g. weeks) in different phases (e.g. requirements specification, design, implementation, testing, and release)? If the processing time for test produces the least amount of requirements per time unit then this is the constraint. As the constraint produces at a slower rate than the other activities inventories that are inputs to the constraint will grow. A method for integrated analysis of multiple inventories is presented in Chapter 7. As in value-stream maps the focus has to be on the end-to-end flow to avoid sub-optimization (LP07) and to achieve a continuous flow of customer value (AP01 - customer satisfaction).

P25: Chief engineer: The chief engineer is one person that is accountable for the success and failure of the development teams. The chief engineer has a wide range of responsibilities (voice of the customer, product concepts and requirements, product architecture and performance, product planning) and requires special abilities (feeling for the needs of customers, exceptional engineering skills, intuition, hard driving teacher and motivator, patient listener, exceptional communicator, etc.) [19]. At the same time the chief engineer does not have formal authorities over the teams. Overall, the chief engineer is an important driving force in optimizing value creation which makes him an initiator to any potential improvement towards agile (AP01 to AP12) or lean (LP01 to LP07).

P26: Kanban pull-system: In a push system requirements are pushed from the requirements activity into the development organization, e.g. by defining a requirements road map and assigning time slots for each requirement. The push approach is likely to result in an overload situation. The Kanban approach, on the other hand, is a pull approach that avoids planning too far ahead (AP02 - welcome change, LP03 - defer commitment). When a work team (or machine in the manufacturing context) has free capacity a signal is given that the next work task can be taken. For software engineering this means that the development systems would pull requirements from a prioritized requirements list when they have enough free capacity. However, teams should not pull new work if there is not enough capacity as this would result in an overload situation creating inventory (LP01 - eliminate waste "partially done work"). Furthermore, if possible they should pull work that has highest priority to the customer (AP01 - customer satisfaction). This strategy might not always be possible due to complex dependencies between features. The management of Kanban is supported by a so-called Kanban board (see Figure 2.3). The Kanban board also shows the progress of development, which fulfills the call of FDD for a progress measure visualizing what has been achieved.

Comparison: Table 2.9 shows the comparison of E2E flow practices. All practices presented in the table are unique to lean as the "see the whole" principle is the driver for their usage. For the value-stream map and the chief engineer all principles of lean and agile are ticked. That does not mean that the approaches guarantee the fulfillment of all principles. However, the value-stream map is a tool that can potentially drive the implementation of practices that fulfill the principles. For example, if the value-stream map helps to discover that long waiting times are due to overload situation then the improvement could be to apply the techniques related to inventory management, as well as to implement a Kanban pull approach. Another discovery through value-streams might be long processing times due to a lack of motivation, which could lead to the practice of teams choosing their own tasks. The chief engineer has all principles ticked as he/she is a potential driver for the improvements by having an overall picture

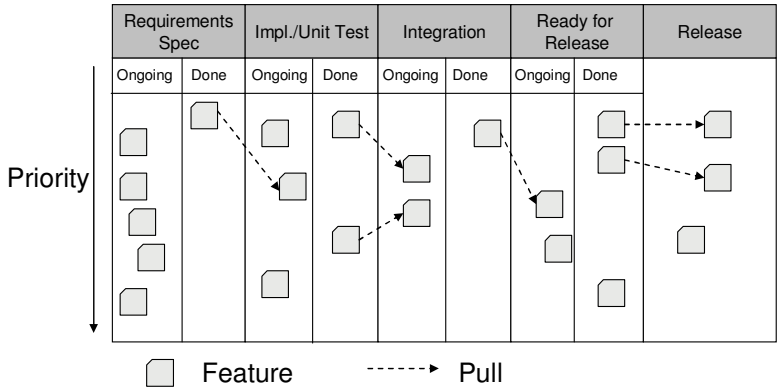


Figure 2.3: Kanban Board

of product development due to the wide range of responsibilities, such as having a customer, architecture, technology, and quality focus.

Table 2.9: Comparison for E2E Flow Practices

	AP01: Customer prio	AP02: Welcome change	AP03: Frequent del.	AP04: Work together	AP05: Motivated individ.	AP06: Face-to-face conv.	AP07: Working software	AP08: Sustainable pace	AP09: Technical excel.	AP10: Simplicity	AP11: Self-org. teams	AP12: Continuous refl.	LP01: Eliminate waste	LP02: Amplify learning	LP03: Defer commit.	LP04: Deliver fast	LP05: Respect people	LP06: Build quality in	LP07: See the whole	Used in agile SE	Used in lean SE
P23: Value-Stream M.	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	
P24: Inventory Mgt.	√		√				√		√				√			√		√	√	√	
P25: Chief Engineer	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	
P26: Pull-Systems	√	√											√		√					√	

Overall Comparison

Based on the previous comparisons we can identify which principles are the same for both paradigms, and which are unique for one of them.

- *Same:* Quality assurance practices and software release practices are the same

for both paradigms. Hence, for these practices lean is agile and agile is lean. All other groups contain practices that are either unique to agile or lean. In total 15 practices are shared between lean and agile.

- *Unique to agile:* The following practices have been identified as unique to agile: on-site customer, coding standards, team-code ownership, planning game, 40 hour week, and stand-up meetings. In consequence for the practices the conclusion is that lean is not agile. However, all practices identified for agile support lean principles and hence are potentially valuable to lean software development.

- *Unique to lean:* One principle in the group "design and implementation" is unique to lean, namely low dependency architecture. Furthermore, all principles connected to the E2E perspective can only be found in the lean context. This is due to the principle "see the whole" that distinguishes lean from agile regarding the principles.

Overall, the analysis shows that lean as well as agile have unique practices (lean has five, and agile has six). Hence, when it comes to practices the statement lean is not agile, and agile is not lean is partially true. The majority of practices are shared between both paradigms. The explanation for the differences can be found in the propositions stated earlier, namely that both paradigms see different ways of implementing their practices, and that lean has a specific focus on E2E which results in unique lean practices.

2.3.4 Processes

The agile development paradigm consists of a number of instantiations in the form of agile processes. The most famous representatives are eXtreme programming (XP) and SCRUM. Both paradigms consist of a set of principles, but also describe a workflow and artifacts that are produced in the process. Detailed descriptions of the processes can be found in [28] for SCRUM and in [3] for XP. However, lean software development does not propose a workflow or the production of specific artifacts. Rather lean states principles and provides analysis tools for processes to guide engineers in improving their processes to achieve a good flow of value. An advantage of not providing a process is that the approaches of lean become more generally applicable, while an agile process often needs tailoring to a specific context. That is, the general agile process models are often not applied exactly as they are described in the books, but are tailored to the specific needs of the context in which they are used (see Chapter 5).

2.4 Discussion

2.4.1 Practical Implications

Potential benefit of E2E perspective in agile: The lean practices (value-stream mapping, inventory management, chief engineer, and pull-systems) are potentially strong tools to support the E2E flow of agile software development. When not having the E2E focus in mind there is, for example, a risk that agile is only implemented in implementation projects and not in the overall development lifecycle. Though, the benefits of agile cannot be leveraged when not focusing on the overall development lifecycle and balancing the flow of work across the complete development lifecycle. Hence, when moving towards agile it is not enough to make the projects agile, as this is likely to result in a sub-optimization of the actual implementation and testing part of development. An additional risk is that the overall flow of requirements is not continuous due to coordination problems when developing large systems with many teams (see Chapter 5). How well the flow actually works can be evaluated with value stream maps and inventory management as shown in [20, 22]. Vice versa, lean could also benefit from the principles unique to agile.

Looking for evidence: Companies are interested in research results that show which practices are most successful in specific contexts. A company wanting to know about the benefits of lean practices should turn to the agile literature due to the large overlap between lean and agile principles. We also have shown that agile principles positively influence the principles of lean as well. Only very little is known about the principles and practices that are related to "see the whole", thus companies looking for information related to principles with an E2E focus will not find much information in the software engineering literature. The high overlap between lean and agile also means that companies having adopted agile practices are not too far away from having a lean software process, given that they add the flow and E2E perspective on-top of their current practices.

2.4.2 Research Implications

Combination of practices: The agile community often states that an agile process only works if all its principles are implemented. However, there is little empirical evidence for this statement. In particular, there is little evidence for which combinations of practices work well in which context (e.g. large-scale vs. small-scale, telecommunication vs. information systems, and so forth). Hence, future research should focus on figuring out which combinations of practices are most beneficial, depending on the context. For example, different practices contribute to the quality related principles. Hence, it

is interesting to know which combination of quality assurance practices is most cost efficient, which calls for further empirical evidence.

Investigate practices related to E2E flow: Several books describe lean practices related to the end to end flow of software development. Though, no research (industrial case studies, experiments, or simulations) document the benefits and challenges related to their usage. For example, further case studies are needed that apply value-stream maps in practice. In addition, little work is done on inventory management and Kanban implementations in software development. We believe that the evaluations will aid in driving the technology transfer of value-stream maps, inventory management, and Kanban.

Overall, lean software development is a research area with great promise and little work done, making it an attractive field for software engineering research.

2.5 Conclusion

This chapter compares two development paradigms (lean and agile development) which are of high relevance for industry due to that they focus on making companies agile in responding to changing market needs. The ability to respond quickly is essential in today's rapidly changing market. The result of the comparison is: (1) Agile and lean agree on the goals they want to achieve; (2) Lean is agile in the sense that the principles of lean reflect the principles of agile, while lean is unique in stressing the end-to-end perspective more; (3) Lean has adopted many practices known in the agile context, while stressing the importance of using practices that are related to the end-to-end flow. These practices are value-stream maps, inventory management, Kanban pull systems, and the use of a chief engineer. In addition, agile uses practices that are not found in lean. The practical implications are that: (1) Industry can potentially benefit from adopting lean practices related to flow, which helped to revolutionize manufacturing and product development; (2) Companies looking for evidence on lean will find important information in the agile literature as there is a high overlap between lean and agile. However, few papers are available that focus on the flow aspect in the software engineering context. The research implications are: (1) Research should focus on investigating which combination of practices should be recommended to industry depending on the industrial context, and (2) practices related to the E2E flow should be investigated to provide evidence for their benefit in the software engineering context.

2.6 References

[1] Bjørn Andersen and Tom Fagerhaug. *Root cause analysis: simplified tools and techniques*. ASQ Quality, Milwaukee, Wis., 2000.

[2] David Anderson. *Agile management for software engineering: applying the theory of constraints for business results*. Prentice Hall, 2003.

[3] Kent Beck. *Extreme Programming explained: embrace change*. Addison-Wesley, Reading, Mass., 2000.

[4] Kent Beck. *Test-driven development: by example*. Addison-Wesley, Boston, MA, 2003.

[5] Kent Beck. *JUnit pocket guide*. O'Reilly, Sebastopol, Calif., 2004.

[6] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in software engineering: A systematic literature review. *Information & Software Technology*, 50(9-10):860–878, 2008.

[7] Tom DeMarco and Timothy Lister. *Peopleware: productive projects and teams*. Dorset House Pub, New York, 2. ed. edition, 1999.

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

[9] Kent Beck et al. Manifesto for agile software development. Web: http://agilemanifesto.org, 2010.

[10] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[11] Michael Hirsch. Moving from a plan driven culture to agile development. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, page 38, 2005.

[12] Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, and Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3):323–332, 2001.

[13] Christer Karlsson and Par Ahlströhm. The difficult path to lean product development. *Journal of Product Innovation Management*, 13(4):283–295, 2009.

[14] Joshua Kerievsky. *Refactoring to patterns*. Addison-Wesley, Boston, 2005.

[15] Alan S. Koch. *Agile software development: evaluating the methods for your organization*. Artech House, Boston, 2005.

[16] Craig Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley, Boston, 2004.

[17] Alexis Leon. *Software configuration management handbook*. Artech House, Boston, Mass., 2. ed. edition, 2005.

[18] Sun Microsystems. Java code conventions. *http://java.sun.com/docs/codeconv*.

[19] James M Morgan and Jeffrey K. Liker. *The Toyota product development system: integrating people, process, and technology*. Productivity Press, New York, 2006.

[20] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *To appear in: Proceedings of the Australian Software Engineering Conference (ASWEC 2010)*, 2010.

[21] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82(9):1479–1490, 2009.

[22] Kai Petersen and Claes Wohlin. Software process improvement through the lean measurement (spi-leam) method. *Journal of Systems and Software, in print*, pages 1479–1490, 2010.

[23] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *Proceedings of the International Conference on Product-Focused Software Process Improvement(PROFES 2009)*, pages 386–400, 2009.

[24] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, Boston, 2003.

[25] Mary Poppendieck and Tom Poppendieck. *Implementing lean software development: from concept to cash*. Addison-Wesley, 2007.

[26] Mary Poppendieck and Tom Poppendieck. *Leading lean software development: results are not the point*. Addison-Wesley, Upper Saddle River, NJ, 2010.

[27] Walter Royce. Managing the development of large software systems: Concepts and techniques. In *Proc. IEEE WESCOM*. IEEE Computer Society Press, 1970.

[28] Ken Schwaber. *Agile project management with Scrum*. Microsoft Press, Redmond, Wash., 2004.

[29] Barry Boehm Stefan Biffl, Aybueke Aurum. *Value-based software engineering*. Springer, New York, NY, 1st ed. edition, 2005.

[30] Gerald M. Weinberg. Egoless programming (excerpt from the psychology of computer programming, silver anniversary edition). *IEEE Software*, 16(1), 1999.

[31] James P. Womack. *The machine that changed the world*. Simon & Schuster, London, 2010.

Chapter 3

The Waterfall Model in Large-Scale Development

Kai Petersen, Claes Wohlin, and Dejan Baca
Published in Proceedings of the International Conference on Product Focused Software Process Improvement (PROFES 2009)

3.1 Introduction

The first publication on the waterfall model is credited to Walter Royce's article in 1970 (cf. [11]). In literature there seems to be an agreement on problems connected to the use of the waterfall model. Problems are (among others) that the model does not cope well with change, generates a lot of rework, and leads to unpredictable software quality due to late testing [13]. Despite the problems identified, the model is still widely used in software industry, some researchers are even convinced that it will be around for a much longer period of time (see [10]). The following trends can be seen in research. First, the model seems to be of very little interest for researchers to focus on as it seems to be old-fashioned. Instead, recent studies have much more focus on agile and incremental development. Secondly, there is very little empirical research backing up what we believe to know about the waterfall model. In order to identify the evidence provided by empirical research on the waterfall model we conducted the following search on Inspec & Compendex:

- ("waterfall model" OR "waterfall development") AND ("empirical" OR "case study" OR "industrial")

Inspec & Compendex was selected as it integrates many full-text databases in computing and thus is considered a good starting point. The search resulted in 33 publications where none of the publications had an explicit focus on studying the waterfall model in an industrial setting. Thus, most of the problems reported on the waterfall model are mainly based on researchers' beliefs and experience reports. Consequently, in order to provide substantial evidence on the usefulness of the waterfall model in industry empirical studies are needed. Evaluating the usefulness empirically aids decision making of whether to use the model in specific context (here large-scale-development).

To address this research gap we conducted a case study focusing on identifying issues in waterfall development and compare them to what has been said in literature. Furthermore, the issues identified are ranked based on their criticality. The case being studied is a development site of Ericsson AB, Sweden. The waterfall model was used at the company for several years. The case study has been conducted according to the guidelines provided by Yin (see [15]). The case study makes the following contributions to research on waterfall development: 1) Illustration of the waterfall implementation in practice within large-scale industrial software development, 2) Identification of issues related to the waterfall model and their prioritization showing the most critical issues, and 3) Comparison of case study results with state of the art (SotA).

The remainder of this chapter is structured as follows: Section 3.2 provides an overview of related work. Thereafter, Section 3.3 illustrates the waterfall model used at the company. Section 3.4 presents the case study design. The analysis of the collected data is provided in Section 3.5 (qualitative analysis) and Section 3.6 (quantitative analysis). Section 3.7 presents a comparison of the case study findings and state of the art. Section 3.8 concludes the chapter.

3.2 Related Work

Literature identifies a number of problems related to the waterfall model. An overview of the problems identified in literature is shown in Table 3.1. In addition to the identified articles we considered books discussing advantages and disadvantages of the waterfall model.

The waterfall model is connected to high costs and efforts [13][8]. That is, it requires approval of many documents, changes are costly to implement, iterations take a lot of effort and rework, and problems are usually pushed to later phases [13]. Few

Table 3.1: Issues in Waterfall Development (State of the Art)

ID	Issue	Reference
L01	High effort and costs for writing and approving documents for each development phase.	[13][8]
L02	Extremely hard to respond to changes.	[13][8][9]
L03	When iterating a phase the iteration takes considerable effort for rework.	[13]
L04	When the system is put to use the customer discovers problems of early phases very late and system does not reflect current requirements.	[11] [13] [4]
L05	Problems of finished phases are left for later phases to solve.	[13]
L06	Management of a large scope of requirements that have to be baselined to continue with development.	[14] [4] [5]
L07	Big-bang integration and test of the whole system in the end of the project can lead to unexpected quality problems, high costs, and schedule overrun.	[6][11][12]
L08	Lack of opportunity for customer to provide feedback on the system.	[6]
L09	The waterfall model increases lead-time due to that large chunks of software artifacts have to be approved at each gate.	[1]

studies are explicitly focused on the waterfall model and some reasons for the failures of the waterfall approach have been identified. One reason mentioned by several studies is the management of a large scope, i.e. requirements cannot be managed well and has been identified as the main reason for failure (cf. [4] [5] [14]). Consequences have been that the customers' current needs are not addressed by the end of the project [4], resulting in that many of the features implemented are not used [5].

Additionally, there is a problem in integrating the overall system in the end and testing it [6]. A survey of 400 waterfall projects has shown that the software being developed is either not deployed or if deployed, it is not used. The reasons for this are the change of needs and the lack of opportunity to clarify misunderstandings. This is caused by the lack of opportunity for the customer to provide feedback on the system [3]. Specifically, the waterfall model fails in the context of large-complex projects or exploratory projects [10].

On the other hand, waterfall development comes with advantages as well. The waterfall model is predictable and pays attention to planning the architecture and structure

of the software system in detail which is especially important when dealing with large systems. Without having focus on architecture planning there is a risk that design decisions are based on tacit knowledge and not explicitly documented and reviewed [2]. Thus, the probability of overlooking architectural problems is high.

3.3 The Waterfall Model at the Company

The waterfall model used at the company runs through the phases requirements engineering, design & implementation, testing, release, and maintenance. Between all phases the documents have to pass a quality check, this approach is referred to as a stage-gate model (see for example [7]). An overview of the process is shown in Figure 3.1.

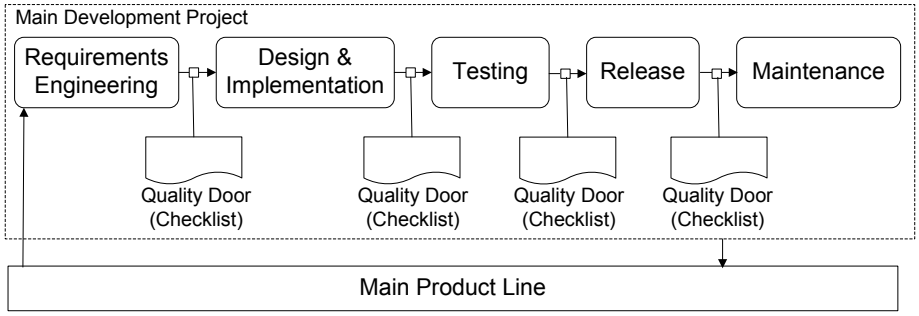


Figure 3.1: Waterfall Development at the Company

We explain the different phases and provide a selection of checklist-items to show what type of quality checks are made in order to decide whether the software artifact developed in a specific development phase can be passed on to the adjacent phase.

Requirements Engineering: In this phase, the needs of the customers are identified and documented on a high abstraction level. Thereafter, the requirements are refined so that they can be used as input to the design and implementation phase. The requirements (on high as well as low abstraction level) are stored in a requirements repository. From this repository, the requirements to be implemented are selected from the repository. The number of requirements selected depends on the available resources for the project. As new products are not built from the scratch, parts from the old product (see main product line in Figure 3.1) are used as input to the requirements phase as well. At the quality gate (among others) it is checked whether all requirements are

understood, agreed upon, and documented. Furthermore, it is checked whether the relevant stakeholders are identified and whether the solution would support the business strategy.

Design and Implementation: In the design phase the architecture of the system is created and documented. Thereafter, the actual development of the system takes place. The developers also conduct basic unit testing before handing the developed code over to the test phase. The quality gate checklist (among others) verifies whether the architecture has been evaluated, whether there are deviations from the requirements compared to the previous quality gate decision, and whether there is a deviation from planned time-line, effort, or product scope.

Testing: In this phase the system integration is tested regarding quality and functional aspects. In order to make a decision whether the the system can be deployed, measures of performance (e.g, throughput) are collected in the test laboratory. As the company provides complete solutions (including hardware and software) the tests have to be conducted on a variety of hardware and software configurations as those differ between customers. The outcome of the phase is reviewed according to a checklist to see whether the system has been verified and whether there are deviations from previous quality gate decisions in terms of quality and time, whether plans for hand-over of the product to the customer are defined according to company guidelines, and whether the outcome of the project meets the customers' requirements.

Release: In the release phase the product is brought into a shippable state. That is, release documentation is finalized (e.g. installation instructions of the system for customers and user-guides). Furthermore, build-instructions for the system have to be programmed. Build-instructions can be used to enable and disable features of the main product line to tailor the system to specific customer needs. At the quality gate (among others) it is checked whether the outcome meets the customers' requirements, whether the customer has accepted the outcome, and whether the final outcome was presented in time and fulfilled its quality requirements. A post-mortem analysis has to be performed as well.

Maintenance: After the product has been released to the customer it has to be maintained. That is, if customers discover problems in the product they report them to the company and get support in solving them. If the problems are due to faults in the product, packages for updating the system are delivered to the customers.

3.4 Case Study Design

The context in which the study is executed is Ericsson AB, a leading and global company offering solutions in the area of telecommunication and multimedia. Such solu-

tions include charging systems for mobile phones, multimedia solutions and network solutions. The company is ISO 2001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the customers will be.

3.4.1 Research Questions

The following main research questions should be answered in the case study:

- *RQ1: What are the most critical problems in waterfall development in large-scale industrial development?*
- *RQ2: What are the differences and similarities between state of the art and the case study results?*

The relevance of the research questions can be underlined as follows: The related work has shown a number of problems related to waterfall development. However, there is too little empirical evidence on the topic and thus more data points are needed. Furthermore, the criticality of problems is not addressed in any way so far, making it hard to decide in which way it is most beneficial to improve the model, or whether the introduction of a new way of working will help in improving the key challenges experienced in the waterfall model.

3.4.2 Case Selection and Units of Analysis

The case being studied is one development site of Ericsson AB. In order to understand the problems that occurred when the waterfall model was used at the company, three subsystems (S1, S2, and S3) are analyzed that have been built according to the model. The systems under investigation in this case study have an overall size of approx. 1,000,000 LOC (as shown in Table 3.2). The LOC measure only includes code produced at the company (excluding third-party libraries). Furthermore, the number of persons involved in building the system are stated. A comparison of the system considered for this study and the size of the Apache web server shows that the system being studied is considerably larger and thus can be considered as large-scale.

3.4.3 Data Collection Procedures

The data is collected through interviews and from process documentation.

Table 3.2: Units of Analysis

	Language	Size (LOC)	No. Persons
Overall System		>5,000,000	-
S1	C++	300,000	43
S2	C++	850,000	53
S3	Java	24,000	17
Apache	C++	220,000	90

Selection of Interviewees

The interviewees were selected so that the overall development life cycle is covered, from requirements to testing and release. Furthermore, each role in the development process should be represented by at least two persons if possible. The selection of interviewees was done as follows:

1. A complete list of people available for the system being studied. Overall 153 people are on this list as shown in Table 3.2.
2. For the selection of persons we used cluster sampling. At least two persons from each role (the roles being the clusters) have been randomly selected from the list. The more persons are available for one role the more persons have been selected.
3. The selected interviewees received an e-mail explaining why they have been selected for the study. Furthermore, the mail contained information of the purpose of the study and an invitation for the interview. Overall, 44 persons have been contacted of which 33 accepted the invitation.

The distribution of people between different roles is shown in Table 3.3. The roles are divided into “What”, “When”, “How”, “Quality Assurance”, and “Life Cycle Management”.

- *What:* This group of people is concerned with the decision of what to develop and includes people from strategic product management, technical managers and system managers.
- *When:* People in this group plan the time-line of software development from a technical and project management perspective.
- *How:* Here, the architecture is defined and the actual implementation of the system takes place. In addition, developers test their own code (unit tests).

- *Quality Assurance:* Quality assurance is responsible for testing the software and reviewing documentation.

- *Life Cycle Management:* This includes all activities supporting the overall development process, like configuration management, maintenance and support, and packaging and shipment of the product.

Table 3.3: Distribution of Interviewees Between Roles and Units of Analysis

	S1	S2	S3	Total
What (Requirements)	2	1	1	4
When (Project Planning)	3	2	1	6
How (Implementation)	3	2	1	6
Quality Assurance	4	3	-	7
Life Cycle Management	6	4	-	10
Total	18	12	3	33

Interview Design

The interview consists of five parts, the duration of the interviews was set to approximately one hour each. In the first part of the interviews the interviewees were provided with an introduction to the purpose of the study and explanation why they have been selected. The second part comprised questions regarding the interviewees background, experience, and current activities. Thereafter, the issues were collected through a semi-structured interview. To collect as many issues as possible the questions have been asked from three perspectives: bottlenecks, rework, and unnecessary work. The interviewees should always state what kind of bottleneck, rework, or unnecessary work they experienced, what caused it, and where it was located in the process.

Process Documentation

Process documentation has been studied to gain an in-depth understanding of the processes. Documentation for example includes process specifications, training material for processes, and presentations given to employees during unit meetings.

3.4.4 Data Analysis Approach

The problems related to the waterfall model at the company have been identified conducting the four steps outlined below. The steps are based on more than 30 hours of interview transcriptions and have been executed by the first author over a three month period.

1. *Clustering:* The raw data from the transcriptions is clustered, grouping statements belonging together. For example, all statements related to requirements engineering are grouped together. Thereafter, statements addressing similar areas within one group (e.g,. all areas that would relate to requirements engineering lead-times) are grouped.

2. *Derivation of Issue Statements:* The raw data contains detailed explanations and therefore is abstracted by deriving problem statements from the raw data, explaining them shortly in one or two sentences. The result was a number of problem statements where statements varied in their abstraction level and could be further clustered.

3. *Mind-Mapping of Issue Statements:* The issue statements were grouped based on their relation to each other and their abstraction level. For example, problems related to requirements lead-times are grouped within one branch called ”long requirements lead-times”. This was documented in form of a mind-map. Issues with higher abstraction level are closer to the center of the mind map than issues with lower abstraction level.

4. *Validation of Issues:* In studies of qualitative nature there is always a risk that the data is biased by the interpretation of the researcher. Therefore, the issues have been validated in two workshops with three representatives from the company. The representatives have an in-depth knowledge of the processes. Together, the steps of analysis described here have been reproduced together with the authors and company representatives. For this a subset of randomly selected issue statements have been selected. No major disagreement has been discovered between the workshop participants on the outcome of the analysis. Thus, the validity of the issue statements can be considered as high.

After having identified the problems they are prioritized into A-problems (critical), B-problems (very important), C-problems (important), D-problems (less important), and E-problems (local). The actual limits on the classes is based on the results. The main objective of the classification is to systematize and structure the data and not to claim that these classes are optimal or suitable for another study.

A. The problem is mentioned by more than one role and more than one subsystem. Moreover, the problem has been referred to by more than 1/3 of the respondents.

B. The problem is mentioned by more than one role and more than one subsystem. Moreover, the problem has been referred to by more than 1/5 of the respondents.

C. The problem is mentioned by more than one role and more than one subsystem. Moreover, the problem has been referred to by more than 1/10 of the respondents.

D. The problem is mentioned by more than one role and more than one subsystem. Moreover, it has been referred to by 1/10 of the respondents or less.

E. The problem is only referred to by one role or one subsystem and thus considered a local or individual problem.

3.4.5 Threats to Validity

Threats to the validity of the outcome of the study are important to consider during the design of the study allowing to take actions mitigating them. Threats to validity in case study research are reported in [15]. The threats relevant to the study are: construct validity, external validity and reliability.

Construct Validity: Construct validity is concerned with obtaining the right measures for the concept being studies. One threat is the selection of people to obtain the appropriate sample for answering the research questions. Therefore, experienced people from the company selected a pool of interviewees as they know the persons and organization best. From this pool the random sample was taken. The selection by the representatives of the company was done having the following aspects in mind: process knowledge, roles, distribution across subsystems, and having a sufficient number of people involved (although balancing against costs). Furthermore, it is a threat that the presence of the researcher influences the outcome of the study. The threat is reduced as there has been a long cooperation between the company and university and the author collecting the data is also employed by the company and not viewed as being external. Construct validity is also threatened if interview questions are misunderstood or misinterpreted. To mitigate the threat pre-tests of the interview have been conducted.

External Validity: External validity is the ability to generalize the findings to a specific context as well as to general process models. One threat to validity is that only one case has been studied. Thus, the context and case have been described in detail which supports the generalization of the problems identified. Furthermore, the process model studied follows the main principles of waterfall development (see Section 3.3)

and thus can be well generalized to that model. In addition, the outcome is compared to state of the art.

Reliability: This threat is concerned with repetition or replication, and in particular that the same result would be found if re-doing the study in the same setting. There is always a risk that the outcome of the study is affected by the interpretation of the researcher. To mitigate this threat, the study has been designed so that data is collected from different sources, i.e. to conduct triangulation to ensure the correctness of the findings. The interviews have been recorded and the correct interpretation of the data has been validated through workshops with representatives of the company.

3.5 Qualitative Data Analysis

In total 38 issues have been identified in the case study. The majority of issues is categorized in class E, i.e, they are only referred to by individuals or are not mentioned across subsystems (see Table 3.4). Furthermore, the distribution of issues between the phases requirements engineering (RE), design and development (DI), verification and validation (VV), release (R), maintenance (M), and project management (PM) is shown. The distribution of issues is further discussed in Section 3.7.

Table 3.4: Number of Issues in Classification

Classification	RE	DI	VV	R	M	PM	No. of Issues
A	1	-	1	-	-	-	2
B	-	-	2	-	-	-	2
C	1	2	-	-	1	1	5
D	1	1	2	-	-	-	4
E	1	1	2	3	8	10	25
Sum	4	4	7	3	9	11	38

In the analysis of the issues we focus on classes A to D as those are the most relevant ones as they are recognized across roles and systems. Thus, they have a visible impact on the overall development process. However, this does not imply that local issues are completely irrelevant, they just have little impact on the overall development process and thus are not recognized by other roles. Table 3.5 shows an overview of the identified issues in classes A to D and their mapping to literature summarized in Table 3.1.

Table 3.5: Issues in Waterfall Development

ID	Class	Process Area	Description	SotA
P01	A	Requirements	Requirements work is wasted as documented and validated requirements have to be discarded or reworked.	L02, L03, L08
P02	A	Verification	Reduction of test coverage due to limited testing time in the end.	L07
P03	B	Verification	Amount of faults found increases with late testing.	L05
P04	B	Verification	Faults found later in the process are hard and expensive to fix.	L07
P05	C	Requirements	Too much documentation is produced in requirements engineering that is not used in later stages of the process.	L01
P06	C	Design	Design has free capacity due to long requirements engineering lead-times.	L09
P07	C	Design	Confusion on who implements which version of the requirements.	-
P08	C	Maintenance	High effort for maintenance (corrections released to the customer).	L04
P09	C	Project Mgt.	Specialized competence focus of team members and lack of confidence.	-
P10	D	Requirements	The impact of requirements on other parts of the system are not foreseen.	L06
P11	D	Design	Design is overloaded with requirements.	-
P12	D	Verification	High amount of testing documentation has to be produced.	L01
P13	D	Verification	Problems in fault localization due to barriers in communication.	-

3.5.1 A Issues

P01: The long lead-times of the requirements engineering phase led to the need to change requirements or discard already implemented and reviewed requirements as the domain investigated (telecommunication) is very dynamic. Furthermore, the distance to the customer caused misunderstandings which resulted in changed requirements or discarded requirements. Due to the complexity of the scope to be defined the num-

ber of requirements was too high for the given resources which resulted in discarding requirements (and sometimes this was done late in the development process). Furthermore, the interviewees emphasized that the decision what is in the scope and what is not takes a lot of time as a high amount of people that have to be involved.

P02: Test coverage in waterfall development was reduced due to multiple reasons. Testing is done late in the project and thus if there have been delays in development, testing has to be compromised as it is one of the last steps in development. Furthermore, too much has to be tested at once after the overall system has been implemented. Additional factors are that testing related to quality is often given low priority in comparison to functional testing, trivial things are tested too intensively, and test resources are used to test the same things twice due to coordination problems.

3.5.2 B Issues

P03: The later the testing, the higher the amount of faults found. The number of faults and quality issues is influenced negatively when using waterfall development. The main cause for this is late testing after everything has been implemented. This provides far too late feedback from test on the software product. Furthermore, basic testing is neglected as there has been low interaction between design and testing, resulting in lack of understanding of each other in terms of consequences of neglecting basic testing. Also due to communication issues, testing started verifying unfinished code which led to a high number of false positives (not real faults).

P04: Having late testing results in faults that are hard to fix, which is especially true for issues related to quality attributes of the system (e.g. performance). These kinds of issues are often rooted in the architecture of the system which is hard to change late in the project.

3.5.3 C Issues

P05: The interviewees emphasized that quite a lot of documentation is produced in the requirements phase. One of the reasons mentioned is limited reuse of documentation (i.e., the same information is reported several times). Furthermore, the concept of quality gates requires producing a lot of documentation and checklists which have to be fulfilled before passing on the requirements to the next phase. Though, in waterfall development the quality gates are required as they assure that the hand-over item is of good enough quality to be used as input for all further development activities.

P06: Design and implementation have free capacity, the reasons being that requirements have to be specified in too much detail, decision making takes a long time, or requirements resources are tied up due to the too large requirements scope. This has a

negative impact on design, as the designers have to wait for input from requirements engineering before they can start working. As one interviewee pointed out *"For such large projects with so many people involved half the workforce ends up working for the rest"*. In consequence, the lead-time of the overall project is prolonged.

P07: From a design perspective, it is not always clear which version of the requirements should be implemented and by whom. The cause of this problem is that work often starts on unfinished or unapproved requirements which have not been properly baselined.

P08: Support is required to release a high number of corrections on already released software. This is due to the overall length of the waterfall projects resulting in very long release cycles. In consequence, the customers cannot wait for the corrections to be fixed for the next release, making corrections a time-pressing issue. Furthermore, the development model requires to handle parallel product branches for customer adaptations of the main product line. In this domain, products have a high degree of variability and thus several product branches have to be supported (see Figure 3.1).

P09: The competence focus of people in waterfall development is narrowed, but specialized. This is due to that people are clearly separated in their phases and disciplines, and that knowledge is not well spread among them. As one interviewee pointed out, there are communication barriers between phases. Furthermore, a lack of confidence has been reported. That is, people are capable but do not recognize their particular strength to a degree they should.

3.5.4 D Issues

P10: New requirements do not have an isolated impact, instead they might affect multiple subsystems. However, due to the large requirements scope, requirements dependencies are often overlooked.

P11: The scope of the requirements was too big for the implementation resources. In consequence, designers and architects were overloaded with requirements which could not be realized with the given resources. Furthermore, after the project has been started more requirements were forced into the project by the customer. In consequence, emergent requirements cannot be implemented by architects and designers as they already face an overload situation.

P12: Test documentation has been done too extensively as the documents became obsolete. The reason for the high amount of documentation was mainly that the process has been very documentation centric.

P13: When dealing with different subsystems, the fault localization is problematic as a problem might only show in one subsystems, but due to communication barriers not all subsystem developers are aware of the problem. In consequence, due to the lack

of communication (see P09) the localization of faults reported by the customer is time consuming.

3.6 Quantitative Data Analysis

Table 3.6 shows the distribution of time (duration) in the development process. The requirements engineering phase takes very long time in comparison to the other phases. The actual implementation of the system seems to be the least time-intensive activity.

Table 3.6: Distribution of Time (Duration) over Phases (in %)

Req.	Impl.&Design	Verification	Release	Total
41	17	19	23	100

Furthermore, we measured the number of change requests per implemented requirement, the discarded requirement, and the percentage of faults found in system test that should have been found in earlier tests (function test and component test). The figures quantify the issues identified earlier. In particular, the high number of discarded requirements and the cause of change requests are related to issue P01. The long lead-times of requirements engineering increase the time-window for change requests and approximately 26 % of all requirements become obsolete. From a quality perspective the fault slip of 31 % is a symptom of P03 (increase of number of faults with late testing) and P04 (the types of faults found in system tests could have been found earlier and thus would have been easier to fix).

Table 3.7: Performance Measures

Measure	Value
CRs per implemented requirement	0.076
Discarded requirements	26 %
Fault slip to system test	31 %

3.7 Comparative Analysis of Case Study and SotA

Table 3.5 relates the issues identified in the case study to the issues mentioned in literature. If an issue from the case study is identified in literature the column SotA provides

the ID of the issue identified in literature (listed in Table 3.1). Through this comparison it becomes apparent that four issues not mentioned in the identified literature have been discovered in the case study, namely P07, P09, P11, and P13. Vice versa all issues acknowledged in literature have been identified in the case study. Table 3.5 also shows that the highest prioritized issues (A and B) have all been mentioned in literature describing the waterfall model. In conclusion researchers and practitioners are aware of the most pressing issues related to waterfall development, while lower prioritized (but still important) issues have not been linked to the waterfall model to the same degree.

The issues in the case study are formulated differently from those identified in literature as the formulation is an outcome of the qualitative data analysis. Therefore, we explain how and why the issues of high priority from the case study and SotA are related to each other. The most critical issues are related to the phases of requirements engineering, and verification and validation (both identified in literature). We found that requirements often have to be reworked and or discarded (P01). The qualitative analysis based on the interviews explained the issue with long lead-times for requirements and large scope making responding to changes hard (related to L02), distance to the customer (related to L08), and change in large scope leads to high effort due to that many people are involved (related to L03). The quantitative analysis shows that 41 % of the lead-time is consumed for requirements engineering. Having to define a large requirements scope extends lead-time and thus reduces requirements stability. In consequence the waterfall model is not suitable in large-scale development in the context of a dynamic market. Regarding verification issue L07 identified in literature states that testing the whole system in the end of the project leads to unexpected quality problems and project overruns. This issue relates to the case study in the following ways: First, testing has to be compromised and thus test coverage is reduced when having fixed deadlines which do not allow for project overruns (P02). Secondly, the faults found late in the process are hard to fix, especially if they are rooted in the architecture of the system (P07).

The issues categorized as C are quite mixed, i.e. they include issues related to requirements, design, maintenance and project management. The issues categorized as D show a similar pattern as the most critical ones (A and B), i.e. they are related to requirements, and verification and validation. Furthermore, one issue is related to design. As mentioned earlier, less than half of the issues classified as C and D have been identified in literature before. An explanation of the issues not yet identified has been provided in the qualitative analysis (see Section 3.5).

It is also interesting to observe that a majority of local issues is related to project management and maintenance (see Table 3.4). Thus, it seems that there is a high number of issues which do not have such an impact on the process that knowledge about them spreads in the organization.

3.8 Conclusion

This case study investigates issues related to the waterfall model applied in the context of large-scale software development and compares the findings with literature. The results are that the most critical issues in waterfall development are related to requirements and verification. In consequence, the waterfall model is not suitable to be used in large-scale development. Therefore, the company moved to an incremental and agile development model in 2005. The comparison of the case study findings with literature shows that all issues found in literature are found in the case study. Though, the case study findings provide more detailed explanations of the issues and identified four new issues, namely 1) confusion of who implements which version of the requirements, 2) high effort for maintenance, 3) specialized competence focus and lack of confidence of people, and 4) problems in fault localization due to communication barriers.

3.9 References

[1] David J. Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results (The Coad Series)*. Prentice Hall PTR, 2003.

[2] Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1):64–69, 2002.

[3] David Cohen, Gary Larson, and Bill Ware. Improving software investments through requirements validation. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop (SEW 2001)*, page 106, Washington, DC, USA, 2001. IEEE Computer Society.

[4] Joe Jarzombek. The 5th annual jaws s3 proceedings, 1999.

[5] Jim Johnson. Keynote speech: Build only the features you need. In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002)*, 2002.

[6] Caspers Jones. *Patterns of Software Systems: Failure and Success*. International Thomson Computer Press, 1995.

[7] Daniel Karlström and Per Runeson. Combining agile methods with stage-gate project management. *IEEE Software*, 22(3):43–49, 2005.

[8] Pete McBreen. *Software craftsmanship : the new imperative*. Addison-Wesley, Boston, 2002.

[9] Shari Lawrence Pfleeger and Joanne M. Atlee. *Software engineering : theory and practice*. Prentice Hall, Upper Saddle River, N.J., 3. ed. edition, 2006.

[10] Lbs Raccoon. Fifty years of progress in software engineering. *SIGSOFT Softw. Eng. Notes*, 22(1):88–104, 1997.

[11] Walter Royce. Managing the development of large software systems: Concepts and techniques. In *Proc. IEEE WESCOM*. IEEE Computer Society Press, 1970.

[12] Johannes Sametinger. *Software engineering with reusable components : with 26 tables*. Springer, Berlin, 1997.

[13] Ian Sommerville. *Software Engineering (7th Edition)*. Pearson Eductation Ltd., 2004.

[14] Michael Thomas. It projects sink or swim. British Computer Society Review 2001, 2001.

[15] Robert K. Yin. *Case Study Research: Design and Methods, 3rd Edition, Applied Social Research Methods Series, Vol. 5*. Prentice Hall, 2002.

Chapter 4

The Effect of Moving from a Plan-Driven to an Incremental and Agile Software Development Approach: An Industrial Case Study

Kai Petersen and Claes Wohlin
Submitted to a Journal

4.1 Introduction

As software has become a major success factor in software products the competition has increased. In consequence, the software industry aims at shorter lead times to gain a first-move advantage and to fulfill the current needs of the customer. However, the needs of the customers in terms of functions and quality constantly evolve leading to high requirements volatility which requires the software companies to be highly flexible. Therefore, more and more software companies started to adopt incremental and agile methods and the number of recent empirical studies on agile methods have

increased (for example [39], [22], and [5]).

Due to the increased importance and interest in agility of software development a systematic review [13] summarized the results of empirical studies on agile methods. According to the systematic review there is a clear need for exploratory qualitative studies. In particular, we need to better understand the impact of the change from traditional (plan-driven) development models (like waterfall, Rational Unified Process (RUP) or V-model) to more agile methods. Furthermore, the review identified research methodological quality problems that frequently occurred in the studies. For example, methods were not well described, the data was biased, and reliability and validity of the results were not always addressed. The review also shows that the main focus of studies was on XP, and that the settings studied are quite small in terms of the number of team members. Overall, this suggests a clear need to further investigate agile and incremental methods using sound empirical methods. Specifically, to understand the impact of migrating to incremental and agile methods requires the comparison of agile with other approaches in different contexts. For example, how does plan-driven development perform in comparison to agile and incremental development in different domains (telecommunication, embedded systems and information systems) and different system complexities (small scale, medium scale, and large scale)?

In order to address this research gap, we conducted a case study investigating the effect of moving from plan-driven development to incremental/agile development. The effect was captured in terms of advantages and issues for the situation before and after the migration. The case being studied was a development site of Ericsson AB, Sweden. The plan-driven approach was used at the company for several years. Due to industry benchmarks and thereby identified performance issues (e.g. related to lead-times) the company first adopted incremental practices starting in the middle of 2005. Agile practices were added in late 2006 and early 2007. Overall, we will show that the company's model shares many practices with incremental development, Extreme Programming (XP), and Scrum.

The case study was conducted in the last quarter of 2007 where incremental practices were adopted to a large part and about 50 % of the Scrum and XP practices have been implemented. We conducted 33 interviews with representatives of different roles in the company to capture the advantages and issues with the two development approaches. That is, we identified issues/advantages in plan-driven development, and how the issues/advantages have changed after migrating to incremental/agile practices. Document analysis was used to complement the interviews. Furthermore, quantitative data collected by the company was used to identify confirmative and contradicting information to the qualitative data. The quantitative data (performance measures) included requirements waste in terms of share of implemented requirements and software quality. The case study research design was strongly inspired by the guidelines pro-

vided in [45]. Furthermore, we used the guidelines provided specifically in a software engineering context by [35].

The contributions of the paper and case study are:

- Illustrate an incremental and agile process model used in industry and comparison of the model with models discussed in literature (e.g., XP, Scrum, and incremental development) to be able to generalize the results.
- Identify and gain an in-depth understanding of the most important issues in relation to process performance in plan-driven development and the incremental/agile process used at the company. The outcomes of the situation before (plan-driven approach) and after the migration (incremental/agile approach) were compared and discussed. This information was captured through interviews, and thus illustrates the perception of the effect of the migration.
- Provide process performance measurements on the development approaches as an additional source of evidence to support or contradict the primary evidence in the form of qualitative findings from the interviews.

The remainder of the paper is structured as follows. Section 4.2 presents related work. Thereafter, Section 4.3 illustrates the development processes used at the company and compares them to known models from literature. Section 4.4 describes the research design. The analysis of the data is divided into one qualitative (Section 4.5) and one quantitative (Section 4.6) part. Based on the analysis, the results are discussed in Section 4.7. Section 4.8 concludes the paper.

4.2 Related Work

Studies have investigated the advantages and disadvantages of plan-driven and agile processes. However, few studies present a comparison of the models in general, and the effect of moving from one model to the other. This section summarizes the results of existing empirical studies on both process models, presenting a list of advantages and disadvantages for each of them. The description of studies related to plan-driven development is not split into advantages and disadvantages as few advantages have been reported in literature.

4.2.1 Plan-Driven Development

Plan-driven development includes development approaches such as the waterfall model, the Rational Unified Process (RUP), and the V-model. All plan-driven approaches

share the following characteristics (cf. [17]): the desired functions / properties of the software need to be specified beforehand; a detailed plan is constructed from the start till the end of the project; requirements are specified in high detail and a rigor change request process is implemented afterwards; the architecture and design specification has to be complete before implementation begins; programming work is only concentrated in the programming phase; testing is done in the end of the project; quality assurance is handled in a formal way.

Waterfall: Challenges with waterfall development (as a representative for plan-driven approaches) have been studied and factors for the failures of the waterfall approach have been identified in empirical research. The main factor identified is the management of a large scope, i.e. requirements cannot be managed well and has been identified as the main reason for failure (cf. [41] [19] [20]). Consequences have been that the customers' current needs are not addressed by the end of the project [19], resulting in that many of the features implemented are not used [20]. Additionally, there is a problem in integrating the overall system in the end and testing it [21]. A study of 400 waterfall projects has shown that only a small portion of the developed code has actually been deployed or used. The reasons for this are the change of needs and the lack of opportunity to clarify misunderstandings. This is caused by the lack of opportunity for the customer to provide feedback on the system [8].

RUP: The RUP process was investigated in a case study mainly based on interviews [15]. The study was conducted in the context of small software companies. The study identified positive as well as negative factors related to the use of RUP. Advantages of the process are: the clear definition of roles; the importance of having a supportive process; good checklists provided by templates and role definitions. Disadvantages of the process are: the process is too extensive for small projects (very high agreement between interviewees); the process is missing a common standard of use; RUP is hard to learn and requires high level of knowledge; a too strong emphasis is put on the programming phase. [16] investigated the effort distribution of different projects using RUP. They found that poor quality in one phase has significant impact on the efforts related to rework in later phases. Thus, balancing effort in a way to avoid poor quality (e.g. more resources in the design phase to avoid quality problems later) is important.

V-model: We were not able to identify industrial case studies focusing on the V-model, though it was part of an experiment comparing different process models (see Section 4.2.3).

Plan-driven approaches are still widely used in practice as recognized in many research articles (c.f. [34] [4] [24] [10]). The case company of this study used the approach till 2005, and there are still new research publications on plan-driven approaches (e.g. [15] [16] [32]).

4.2.2 Incremental and Agile Development

[13] conducted an exhaustive systematic review on agile practices and identified a set of relevant literature describing the limitations and benefits of using agile methods. According to the systematic review a majority of the relevant related work focuses on XP (76 % of 36 relevant articles). The following positive and negative factors have been identified in the review.

Positive factors: Agile methods help to facilitate better communication and feedback due to small iterations and customer interaction (cf. [39] [22] [2]). Furthermore, the benefit of communication helps to transfer knowledge [2]. Agile methods further propose to have the customer on-site. This is perceived as valuable by developers as they can get frequent feedback [40] [39] [22], and the customers appreciate being on-site as this provides them with control over processes and projects [18]. An additional benefit is the regular feedback on development progress provided to customers [18]. From a work-environment perspective agile projects are perceived as comfortable as they can be characterized as respectful, trustful, and help preserving quality of working life [26].

Negative factors: Well known problems are that architecture does not have enough focus in agile development (cf. [28] [38]) and that agile development does not scale well [7]. An important concept is continuous testing and integration. Though, realizing continuous testing requires much effort as creating an integrated test environment is hard for different platforms and system dependencies [39]. Furthermore, testing is a bottleneck in agile projects for safety critical systems, the reason being that testing had to be done very often and at the same time exhaustively due to that a safety critical system was developed [43]. On the team level team members have to be highly qualified [29]. With regard to on-site customers a few advantages have been mentioned. The downside for on-site customers is that they have to commit for the whole development process which requires their commitment over a long time period and puts them under stress [27].

Petersen and Wohlin (cf. [30]) compared issues and advantages identified in literature with an industrial case (see Chapter 5). The source of the information was the interviews conducted in this study, but the paper focused on a detailed analysis of the agile situation, and its comparison with the literature. The main finding was that agile practices lead to advantages in one part of the development process, and at the same time raises new challenges and issues in another part. Furthermore, the need for a research framework for agile methods has been identified to describe the context and characteristics of the processes studied.

4.2.3 Empirical Studies on Comparison of Models

In waterfall development the requirements are specified upfront, even the requirements that are not implemented later (due to change). The introduction of an incremental approach reduces the impact of change requests on a project. Furthermore, the increments can be delivered to the customer more frequently demonstrating what has been achieved. This also makes the value of the product visible to the customer early in development [9]. Furthermore, several studies indicate that agile companies are more customer centric and generally have better relationships to their customers. This has a positive impact on customer satisfaction [6] [37]. However, a drawback of agile development is that team members are not as easily interchangeable as in waterfall-oriented development [3].

Given the results of the related work it becomes apparent that benefits reported were not identified starting from a baseline, i.e. the situation before the introduction of agile was not clear. Hence, little is known about the effect of moving from a plan-driven to an incremental and agile approach. Furthermore, the focus of studies has been on eXtreme programming (XP) and the rigor of the studies was considered as very low [13]. Hence, the related work strengthens the need for further empirical studies investigating incremental and agile software development. Furthermore, evaluating the baseline situation is important to judge the improvements achieved through the migration. In response to the research gap this study investigates the baseline situation to judge the effect of the migration towards the incremental and agile approach.

4.3 The Plan-Driven and Agile Models at the Company

Before presenting the actual case study the process models that are compared with each other have to be introduced and understood first.

4.3.1 Plan-Driven Approach

The plan-driven model that was used at the company implemented the main characteristics of plan-driven approaches as summarized by [17]. The main process steps were requirements engineering, design and implementation, testing, release, and maintenance. At each step a state-gate model was used to assure the quality of the software artifacts passed on to the next phase, i.e. software artifacts produced have to pass through a quality door. The gathered customers' needs collected from the market by so-called market units were on a high abstraction level and therefore needed to be specified in detail to be used as input to design and development. Requirements were

stored in a requirements repository. From the repository, requirements were selected that should be implemented in a main project. Such a project lasted from one up to two years and ended with the completion of one major release. Quality checks related to the requirements phase were whether the requirements have been understood, agreed on, and documented. In addition it was determined whether the product scope adhered to the business strategy, and whether the relevant stakeholders for the requirements were identified. The architecture design and the implementation of the source code was subjected to a quality check with regard to architecture evaluation and adherence to specification, and whether the time-line and effort deviated from the targets. In the testing phase the quality door determined whether the functional and quality requirements have been fulfilled in the test (e.g. performance, load balancing, installation and stability). It was also checked whether the hand-over of the product to the customer was defined according to company guidelines. In the release phase the product was packaged, which included programming of build instructions. They were used to enable and disable features to be able to tailor the system to specific customer needs. The documentation also contains whether the customer accepted the outcome, and whether the final result was delivered meeting the time and effort restrictions.

When changes occured in form of a change request (CR), requirements had to be changed and thus became obsolete. Therefore, all downstream work products related to these requirements, like design or already implemented code, had to be changed as well. Late in the process, this led to a considerable amount of rework [42] and prolonged lead-times. Furthermore, software development has not only to cope with changes in needs that are valid for the whole customer base, but also with customer specific needs. If the customer specific needs were considered as of high priority, a customer adaptation project was initiated which took the last available version of the product as input. In response to these challenges the company recognized the need for a more agile and flexible process leading to the stepwise introduction of incremental and agile practices, as described in the following subsections.

Further details on the plan-driven approach employed at the company (i.e. the baseline situation) can be found in Chapter 3.

4.3.2 Agile and Incremental Model

The agile and incremental process model used at the company is shown in Figure 4.1. The process relied on a set of company specific practices that have been introduced. The numbers (1 to 5) in Figure 4.1 map to the enumeration of the following practices:

1. *Product Backlog:* The packaging of requirements for projects was driven by requirement priorities. Requirements with the highest priorities were selected and

packaged to be implemented. Another criterion for the selection of requirements was that they fit well together and thus could be implemented in one coherent project.

2. *Anatomy Plan:* Furthermore, an anatomy plan was created, based on the dependencies between the parts of the system being implemented in each project. The dependencies were a result of system architecture, technical issues and requirements dependencies. The anatomy plan resulted in a number of baselines called latest system versions (LSV) that needed to be developed. It also determined the content of each LSV and the point in time when a LSV was supposed to be completed. The anatomy plan captured dependencies between features (e.g. one feature had to be ready before another one was implemented) and technical dependencies. Technical dependencies are critical in the telecommunication domain as platforms and communication protocols change. For example, if a version of the software ran on one protocol version it could not be integrated with the new protocol version. Therefore, besides the prioritization in the product backlog the anatomy plan provided important input on the order in which projects were run, and when increments could be integrated and tested.

3. *Small Teams and Time-line:* The requirements packages were implemented by small teams in short projects lasting approximately three month. The duration of the project determined the number of requirements selected for a requirement package. Each project included all phases of development, from pre-study to testing. As emphasized in the figure, when planning the order in which the projects were executed the prioritization as well as technical dependencies on the architecture level had to be taken into consideration. Furthermore, the figure shows that an interaction between requirements and architecture took place.

4. *Use of Latest System Version:* If a project was integrated with the last baseline of the system, a new baseline was created (referred to as LSV). Therefore, only one baseline existed at one point in time, helping to reduce the effort for product maintenance. The LSV can also be considered as a container where the results of the projects (including software and documentation) are put together. When the results of the projects had been integrated a system test took place in the LSV, referred to as LSV test. When in time a test should be conducted was defined by testing cycles and for each testing cycle it was defined which projects should drop within the next cycle. Comparing the work done on team level with the work done in the LSV one can say that on the project level the goal was to focus on the development of the requirements packages while the LSV focused

on the overall system where the results of the projects were integrated. With the completion of the LSV the system was ready for release.

5. *Decoupling Development from Customer Release:* If every release would have been pushed on the market, there would be too many releases in use by customers needing support. In order to avoid this, not every LSV was to be released, but it had to be of sufficient quality to be possible to release to customers. LSVs not released to the customer were referred to as potential releases (see practice 5 in Figure 4.1). The release project in itself was responsible for making the product commercially available and to package it in the way that the system could be released.

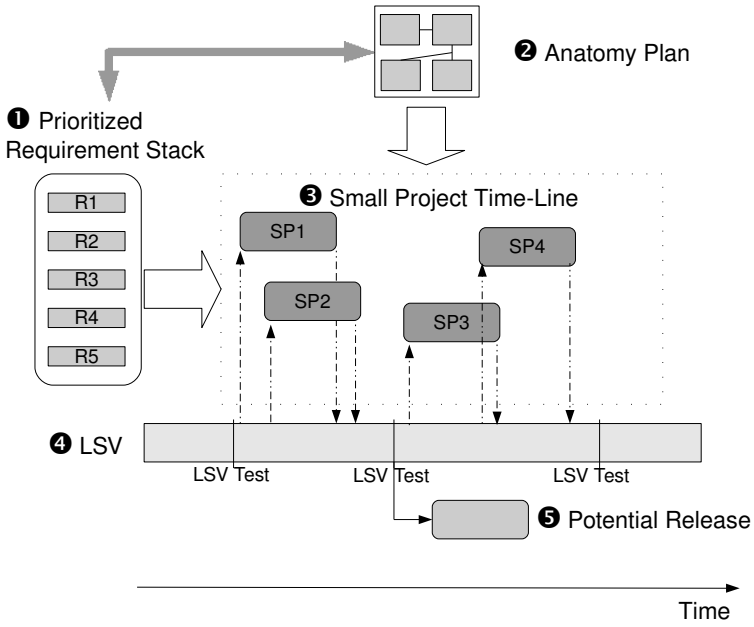


Figure 4.1: Development Process

The transition from the plan-driven to the incremental and agile approach was done step-wise. The implementation of the incremental process formed the basis for agile software development. Therefore, it was essential to establish the practices small teams, LSV, and product backlog together in the first step. This enabled the teams to

deliver continuously from a product backlog towards a baseline for testing (LSV). With this basic process in place the second step could be implemented, i.e. the teams moving towards an agile way of working through continuous reflection and improvement, and frequent face to face interactions through stand-up meetings. Furthermore, the introduction of the last system version optional releases were enabled. In the future the company plans to further extend the agile way of working by introducing additional practices, such as test driven development, requirements formulated as user stories, refactoring, low dependency architecture,

4.3.3 Comparison with General Process Models

The company's process model was created based on practices applied in general incremental and agile process models. To be able to generalize the results of this study, the characteristics of the incremental and agile model used at the company (C) were mapped to the existing models of incremental and iterative development (ID), Extreme programming (XP), and Scrum (SC). That is, if the application of a specific practice leads to problems in the model investigated in this case study, it might also cause problems in models applying the same principle. Table 4.1 (created based on the information provided in [25]) shows that 4 out of 5 incremental principles are fulfilled which means that lessons learned in this study are generalizable to ID. Furthermore, the model used at Ericsson shares 5 out of 12 principles with XP and 6 out of 10 principles with Scrum.

The company's model realizes the principles shared with ID, XP and Scrum as follows:

- *Iterations and Increments:* Each new LSV was an increment of the product. Projects were conducted in an iterative manner where a set of the projects' increments was dropped to the LSV.
- *Internal and External Releases:* Software products delivered and tested in the LSV could be potentially delivered to the market. Instead of delivering to the market, they could also be used as an input to the next internally or externally used increment.
- *Time Boxing:* Time boxing means that projects have a pre-defined duration with a fixed deadline. In the company's model the time box was set to approximately three month. Furthermore, the LSV cycles determined when a project had to finish and drop its components to the LSV.
- *No Change to Started Projects:* If a feature was selected and the implementation realizing the feature has been started then it was completed.

Table 4.1: Comparison with General Process Models

Principle	ID	XP	SC	C
Iterations and Increments	√	√	√	√
Internal and External Releases	√			√
Time Boxing	√	√	√	√
No Change of Started Projects	√		√	√
Incremental Deliveries	√			√
On-site Customer		√	√	
Frequent Face-to-Face Interaction		√	√	√
Self-organizing Teams		√	√	
Empirical Process		√	√	
Sustainable Discipline		√		
Flexible Product Backlog		√	√	√
Fast decision making			√	
Frequent Integration		√	√	√
Simplicity of design		√		
Refactoring		√		
Team Code Ownership		√		

- *Frequent Face-to-Face Interaction:* Projects were realized in small teams sitting together, the teams consisting of six or seven persons including the team leader. Each team consisted of people fulfilling different roles. Furthermore, frequent team meetings were conducted in the form of stand-up meetings as used in Scrum.

- *Product Backlog:* A prioritized requirements list where the highest prioritized requirements were taken from the top and implemented first was one of the core principles of company's model of development. The product backlog could be continuously re-prioritized based on market-changes allowing for flexibility.

- *Frequent Integration:* Within each LSV cycle the results from different projects were integrated and tested. As the cycles have fixed time frames frequent integration was assured.

Overall it was visible that the model shares nearly all principles with ID and realizes a large portion of XP and Scrum principles. Agile software development literature points to why the principles used at Ericsson should increase the agility, i.e. the ability

of the company to respond to changing requirements. The main source of agility was the prioritized requirements list, which was very similar to the flexible product backlog in Scrum [36]. Hence, the development was flexible when the needs of the customers change as the backlog was continuously re-prioritized. Furthermore, new features were selected from the backlog continuously and are integrated frequently, which means that one can deliver less functionality more frequently, which provides flexibility and the opportunity for adaptive planning [36, 25, 23]. This is very much in line with agile saying that the primary measure of progress is working software and that the software should be useful [25]. In contrast, waterfall development would define the whole requirements list upfront and integrate the implementation in the end and hence working software would not be produced continuously [32]. Consequently requirements become obsolete as they are only delivered together creating very long lead-times. The need for change in the backlog was communicated through market units as the process was market-driven without a specific customer, but a large number of potential customers. Requirements engineers and system experts then discuss the change that is needed. The primary method for prioritizing the requirements was to have a ranked list. We acknowledge that not all agile practices of a specific model were fulfilled. However, due to the specific nature of the development at Ericsson (market-driven with unknown customers and large-scale products) the practitioners made the decision to select practices they considered to be most beneficial in their specific context.

4.4 Case Study Design

4.4.1 Study Context

It is of importance to describe the context in order to aid in the generalizability of the study results (cf. [31]). Ericsson is one of the major telecommunication companies in the world offering products and services in this domain including charging solutions for mobile phones, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The development of the company is market-driven and characterized by a frequently changing market. Furthermore, the market demands highly customized solutions (for example customizations for specific countries). Further details regarding the context of the study are shown in Table 4.2.

4.4.2 Research Questions and Propositions

In this study, we aimed at answering the following research questions:

- *RQ1 What issues in terms of bottlenecks, unnecessary work, and rework were perceived before and after the migration from plan-driven to incremental and agile development?* The first research question is the basis for further improvement of the process models.
- *RQ2: How commonly perceived are the issues (bottlenecks, unnecessary work, and rework) for the each of the development approaches and in comparison to each other?* The second research questions aims at capturing the effect of the change from plan-driven to incremental/agile development by determining the change in how commonly perceived the issues were in each of the approaches.
- *RQ3: Does the quantitative performance data (requirements waste and data on software quality) support or contradict the qualitative findings in RQ1 or RQ2?* Collecting these measures provides quantitative measures on the actual change in process performance at the company, thus being able to serve as an additional source of evidence as support for the qualitative analysis.

Based on the research questions, research propositions are formulated. Study propositions point the researcher into a direction where to look for evidence in order to answer the research questions of the case study [45]. A proposition is similar to a hypotheses, stating what the expecting outcome of the study is. The following propositions are made for this case study:

- *Proposition 1 (related to RQ1): Different issues are mentioned by the interviewees for the process models.* Literature reports problems specific for plan-driven and agile development (see Section 4.2). Thus, we assume to also find different problems before and after the migration.
- *Proposition 2 (related to RQ2 and RQ3): The qualitative and quantitative data shows improvements when using agile and incremental practices*. The agile and incremental model used at the company was specifically designed to avoid problems that the organization was facing when using a plan-driven approach. For example, too long durations in the requirements phase leading to a vast amount of requirements changes prior to development. Therefore, we hypothesize that 1) the issues raised for the incremental/agile way of working are less commonly perceived than those raised for the plan-driven approach, and 2) there is an improvement regarding performance measures with the new incremental and agile approach.

In order to answer the research questions and evaluate the propositions, one of Ericsson's development sites was selected as a case. The case and units of analysis are described in more detail in the following section.

Table 4.2: Context Elements

Context Element	Description
Maturity	All systems older than 5 years
Size	Large-scale system with more than 5,000,000 LOC overall
Domain	Telecommunication and multimedia solution
Market	Highly dynamic and customized market
Process	On the principle level incremental process with agile practices in development teams
Certification	ISO 9001:2000
Requirements engineering	Market-driven process, i.e. requirements were collected by market units from large customer base. Actual customers that will buy the product are to a large extent unknown while developing. Requirements handed over to development unit and were available to development and implementation in form of a prioritized backlog.
Requirements documentation	Requirements written in natural language on two abstractions, high level requirements referred to as quick studies and detailed requirements for development teams.
Requirements tracking	Requirements proprietary tool for managing requirements on product level (i.e. across projects). Requirements database can be searched and requirements have been tagged with multiple attributes (e.g. source, target release)
Practices	Iterations and increments, internal and external releases, time boxing, no change of started projects, frequent face to face interaction, product backlog, frequent integration (see Table 4.1)
Agile maturity	Stepwise implementation of agile started in 2005.
Testing practices and tools	Unit and component test (Tools: Purify, JUnit), Application and integration test verifying if components work together (JUnit, TTCN3), LSV test verifying load and stability, load balancing, stability and upgradability, compatibility, and security (TTCN3). Unit tests were conducted by the persons writing the code to be unit tested, while the LSV test is done by testing experts.
Defect tracking	Company-proprietary tool capturing where defects were found and should have been found, status in defect analysis process, etc.
Team-size	Six to seven team members.
Size of development unit	Approx. 500 people in research and development.
Distribution	Systems investigated were developed locally.

4.4.3 Case Selection and Units of Analysis

The case selection allows to gain insights into issues related to process performance in the situation where a large scale system is developed within a frequently changing

environment. This can be used as input to identify issues that need to be addressed in large scale development to develop a flexible process, as flexibility and short time to market are essential requirements posed on the process in our study context.

As the process models presented earlier were used company-wide, the processes investigated can be considered as representative for the whole development site as well as company-wide. Within the studied system, three different subsystem components were studied which represent the units of analysis (subsystem 1 to 3). A subsystem was a large system component of the overall system. Table 4.3 provides information of the system complexity in lines of code (LOC) and number of persons involved. The LOC measure only included code produced at the company (i.e., third-party frameworks and libraries are excluded). Furthermore, as a comparison to the Ericsson systems, the size measure in LOC for the open source product Apache web server (largest web server available) is shown as well, the LOC being counted in the same way.

Table 4.3: Units of Analysis

	Language	Size (LOC)	No. Persons
Overall System		>5,000,000	-
Subsystem 1	C++	300,000	43
Subsystem 2	C++	850,000	53
Subsystem 3	Java	24,000	17
Apache	C++	220,000	90

The figures show that the systems were quite large, all together more than 20 times larger than the Apache web server. To study the processes of the subsystems, a number of people were interviewed and the measures for each subsystem were collected. The distribution of interviewees and the data collection procedures are explained in the following.

4.4.4 Data Collection Procedures

The data was collected from different sources, following the approach of triangulation. The first source driving the qualitative analysis was a set of interviews. The second source was process documentation and presentations on the progress of introducing incremental and agile practices. The third source were performance measures collected by the company. This section explains the data collection procedures for each source in detail.

Selection of Interviewees

The interviewees were selected so that the overall development life cycle were covered, from requirements to testing and product packaging. Furthermore, each role in the development process should be represented by at least two persons if possible. That is, these persons fill out the role as their primary responsibility. Only interviewees with process experience were selected. Prior to the main part of the interview the interviewees were asked regarding their experience. We asked for the duration the interviewees have been working at the company, and the experience with the old process model (plan-driven) and the new process model (incremental and agile). The experience was captured by asking for activities that support good knowledge with regard to the process model, such as study of documentation, discussion with colleagues, seminar and workshops, and the actual use in one or more projects. The average duration of the interviewees working at the studied company was 9.4 years. Only two persons interviewed worked less than two years at the company. Ten persons had at least 10 years of experience working at the company. This indicates that the interviewees had very good knowledge of the domain and the company's processes. They were very familiar with the old process model with regard to all learning activities mentioned before. With regard to the new process model trainings have been given to the all interviewees. In addition, the new process model was widely discussed in the corridors which supported the spread of knowledge about it. Eighteen of the interviewees already completed at least one project with the new development approach, for the remaining interviewees projects using the new approach were currently ongoing, i.e. they were in a transition phase. Overall, the result of the experience questionnaire showed good knowledge and awareness of the processes, which was also visible in their answers.

The selection process of interviewees was done using the following steps:

1. A complete list of people available for each subsystem was provided by management, not including newly employed personal not familiar with the processes.

2. At least two persons from each role were randomly selected from the list. The more persons were available for one role the more persons were selected.

3. The selected interviewees received an e-mail explaining why they had been selected for the study. Furthermore, the mail contained information of the purpose of the study and an invitation for the interview. Overall, 44 persons had been contacted of which 33 accepted the invitation.

The distribution of people between different primary responsibilities and the three subsystems (S1-S3) is shown in Table 4.4. The roles are divided into *What*, *When*, *How*, *Quality Assurance*, and *Life Cycle Management*.

Table 4.4: Distribution of Interviewees Between Roles and Units of Analysis

	S1	S2	S3	Total
What (Requirements)	2	1	1	4
When (Project Planning)	3	2	1	6
How (Implementation)	3	2	1	6
Quality Assurance	4	3	-	7
Life Cycle Management	6	4	-	10
Total	18	12	3	33

- *What:* This group is concerned with the decision of what to develop and includes people from strategic product management, technical managers and system managers. Their responsibility is to document high-level requirements and detailing them for design and development. Roles involved in this group are product managers and system managers specifying detailed requirements.
- *When:* People in this group plan the time-line of software development from both technical and project management perspectives. This includes system managers being aware of the anatomy plan, as well as line and project managers who have to commit resources.
- *How:* Here, the architecture is defined and the actual implementation of the system takes place. Developers writing code also unit test their code.
- *Quality Assurance:* Quality assurance is responsible for testing the software and reviewing documentation. This group primarily contains expert testers having responsibility for the LSV.
- *Life Cycle Management:* This includes all activities supporting the overall development process, like configuration management, maintenance and support, and packaging and making the product available on the market.

Interview Design

The design of the interview consisted of five parts, the duration of the interviews was one hour. In the first part of the interview, an introduction of the study's goals was provided. Furthermore, the interviewees were informed why they had been selected for the interview. It was also made clear that they were selected randomly from a list

of people, and that everything they say would be treated confidentially. In the second part, the interviewees were asked for their experience and background regarding work at the company in general, and experience with the plan-driven and incremental/agile development approaches in particular. Therefore, the interviewees filled in a questionnaire rating their experience with the two process models. Thereafter, the actual issues were collected through a semi-structured interview, asking for issues that could be characterized as bottlenecks, avoidable rework and unnecessary work (for descriptions see Table 4.5). Asking for those areas stimulated the discussion by helping the interviewee to look at issues from different perspectives and thus allowing to collect many relevant issues. We asked for issues regarding the plan-driven approach and the incremental/agile approach.

Table 4.5: Questions for Issue Elicitation

Area	Description
Bottlenecks	Bottlenecks are single components hindering the performance of the overall development process. A cause for a bottleneck is the low capacity offered by the component [1].
Unnecessary Work	We understand unnecessary work as activities that do not contribute to the creation of customer value. In lean development, this is referred to as producing waste ([1] [33]).
Avoidable Rework	Rework can be avoided when doing things completely, consistently and correctly [14]. For example, having the right test strategy to discover faults as early as possible ([12] [11]).

The interviewees should always state the cause of the issue and where the symptoms of the issue became visible in the process. During the course of the interview, follow-up questions were asked when interesting issues surfaces during the course of the interview. All interviews were recorded and transcribed. The interview protocol can be found in Appendix A.

Process Documentation

The company provided process documentation to their employees, as well as presentations on the process for training purposes. This documentation was used to facilitate a good understanding of the process in the organization (see Section 4.3). Furthermore, presentations given at meetings were collected which showed the progress and first results of the introduction of incremental and agile practices from the management perspective. In addition to that, the documentation provided information of problems with plan-driven development, which led the company to the decision of making the transition to incremental and agile development. Overall the documentation served two main purposes: (1) Help the interviewer to gain an initial understanding of how the processes work prior to the interview. In addition, the interviewer needed to become familiar with the terminology used at the company, which was also well supported by documentation; (2) To extract context information relevant for this study.

Performance Measures

The company collected a number of performance measures on their development processes and projects. The performance measures were identified at the company to provide an indication of performance changes after moving to incremental and agile development. The measurements were selected based on availability and usefulness for this study.

- *Requirements waste and change requests:* Requirements waste means that requirements are elicited, documented and verified, but they are not implemented. The analysis focused on the ratio of implemented requirements in comparison to wasted requirements. Furthermore, the change requests per requirement were analyzed. Requirements waste and change requests indicate whether the company increases its ability to develop requirements in a timely manner after the customer need was raised. If there are fewer change requests and less discarded requirements then this is an indicator for that the current market needs are fulfilled in a better way. The information for waste and change requests was attributed to the plan-driven and incremental development model through releases, i.e. it was known which releases used the purely plan-driven process, and which releases used the new incremental process with additional agile practices.

- *Quality Data:* The change in software quality was analyzed through fault-slip through and maintenance effort. Fault-slip-through [12] shows how many faults were identified in the LSV which should have been found earlier. In order to be able to measure the fault-slip-through a testing strategy has to be developed. The

strategy needs to document which type of fault should be detected in a specific phase (e.g. performance related issues should be detected in system test, buffer overflows should be detected in unit testing and static code analysis, etc.) and when they were actually detected. That way one can determine how many faults should have been detected before a specific quality assurance phase. In this case study the quality of basic test and function test conducted before integration and system test was measured. For example, a fault-slip of x% in the system testing phase means that x% of all faults discovered in this phase should have been found in earlier phases (e.g. function testing). The data source for the fault-slip through measurements was the defect tracking system employed at the company, which was introduced in Table 4.2. The maintenance effort was a an indicator of the overall quality of the product released on the market. Quality data was considered as quality is an important aspect of the market Ericsson operates in. For telecommunication operators performance and availability are particularly important quality characteristics.

4.4.5 Data Analysis

The data analysis was done in six different steps, as shown in Figure 4.2. The first four activities led to a set of issues related to process performance in both development approaches. The first author transcribed all interviews resulting in more than 30 hours of interview data. Thereafter, the author conducted the first four steps over a three month period based on the transcriptions.

1. *Clustering of Raw Data:* The statements from each interviewee were mapped to process phases, the role of the interviewee, and the process model they refer to (i.e. either plan-driven or incremental/agile development). The information was maintained using a matrix. For each statement the identity-number of the interviewee was documented as well to assure traceability.

2. *Derivation of Issues from Raw Data:* As the raw data contained detailed explanations using company specific terminology the data was summarized and reformulated by deriving issues from the clustered data. Each issue was shortly described in one or two sentences. The result was a quite high number of issues in each group, as the issues were on different abstraction levels.

3. *Mapping of Issues:* The issues were grouped based on their relation to each other, and their abstraction level. For example, issues that negatively affect the coverage of the system by test cases were grouped within one branch called “low test coverage”. The grouping was documented in the form of a mind map. Issues

with higher abstraction level were closer to the center of the mind map than issues with lower abstraction level.

4. *Issue Summary and Comparison:* The issues on the highest abstraction level were summarized in the form of short statements and used for further analysis (as presented in the Sections 4.5 and 4.6).

An example of the analysis steps is illustrated in Appendix B.

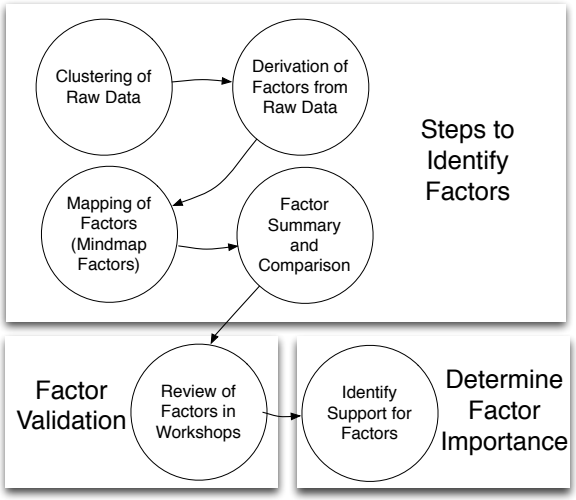


Figure 4.2: Data Analysis Process for Qualitative Data

Furthermore, the last two activities were concerned with validating the list of issues and determining the importance of the issues within the organization.

5. *Validation of Issues:* The fifth step was the validation of the derived issues. The authors and three representatives from the company participated in a workshop to review the issues. All representatives from the company had an in-depth knowledge of both process models. The validation was done by randomly selecting issues and each of the representatives of the company reviewed the steps of issue derivation outlined before. There was no disagreement on the interpretation of the raw data and the issues derived. Furthermore, all participants of the workshop reviewed the final list of issues, only having small improvement suggestions on how to formulate the issues. That is, the list of issues could be considered of high quality.

6. *Weight of Issues:* The sixth step aimed at identifying the most commonly perceived issues with regard to the approaches (plan-driven and incremental/ agile). As we asked the interviewees to state three bottlenecks/ unnecessary works/ reworks for each of the models we were able to determine which issues mentioned were most commonly perceived. For example, if an interviewee an issue as critical for plan-driven, but not for incremental and agile, this is an indication for an improvement of the issue. We explicitly asked the interviewees for the situation before and after the migration, and used follow-up questions whenever it was unclear whether an issue was only considered important for one of the process models. In order to determine which issues were the most common, the data was first divided into global and local issues. The division in global and local issues was defined as follows:

   - *Global Issues:* Global issues were stated by interviewees representing more than one role and representing more than one subsystem component (i.e., they were spread across the units of analysis).
   - *Local Issues:* Local issues were stated by one or several interviewees representing one role or one subsystem component.

   To systematize the global issues, four different subgroups were defined. The main objective of the grouping was to structure the responses based on the number of interviewees mentioning each issue. It was hence a way of assigning some weight to each issue based on the responses. The following four subgroups were defined:

   - *General Issues:* More than 1/3 of the interviewees mentioned the issue.
   - *Very Common Issues:* More than 1/5 of the interviewees mentioned the issue.
   - *Common Issues:* More than 1/10 of the interviewees mentioned the issue.
   - *Other Issues:* Less than 1/10 of the interviewees mentioned the issue, but it was still mentioned by more than one person representing different roles or different subsystem components.

   In addition to that, the interviewees explicitly talked about inferences with regard to improvements that they have recognized after moving to incremental and agile development. The improvements were grouped into commonly perceived and observation. One should observe that the threshold for commonly perceived improvements was much lower compared to the above thresholds, and fewer groups

were formulated. This was due to that we did not explicitly ask for the improvements as the interviews focused on issues determining how the commonality of issues changed after migration. However, in several cases the interviewee also talked about the actual differences between the situation before and after the migration and hence the improvements perceived when moving from a plan-driven approach to incremental and agile development. Thus, the improvements perceived were only divided into two groups:

- *Commonly perceived:* More than 1/10 of the interviewees representing more than one subsystem component mentioned the issue.
- *Observation:* Less than 1/10 of the interviewees mentioned the issue.

It should be observed that local issues also could be of high importance. However, they may be perceived as local since the issue is not visible outside a certain phase, although major problems inside a phase were communicated to others. Thus, it is believed that the main issues influencing process performance are captured in the global issues.

4.4.6 Threats to Validity

Research based on empirical studies does have threats, and hence so does the case study in this paper. However, the success of an empirical study is to a large extent based on early identification of threats and hence allowing for actions to be taken to mitigate or at least minimize the threats to the findings. Threats to case studies can be found in for example [45], and threats in a software engineering context is discussed in for example [44]. The threats to validity can be divided into four types: construct validity, internal validity, external validity and reliability (or conclusion validity). Construct validity is concerned with obtaining the right measures for the concept being studies. Internal validity is primarily for explanatory and causal studies, where the objective is to establish a causal relationship. External validity is about generalizability to determine to which context the findings in a study can be generalized. Finally, reliability is concerned with repetition or replication, and in particular that the same result would be found if re-doing the study in the same setting.

Construct Validity

The following threats were identified and the corresponding actions were taken:

- *Selection of people:* The results are highly dependent on the people being interviewed. To obtain the best possible sample, the selection of people was done by

people having worked at Ericsson for a long time and hence knowing people in the organization very well.

- *Reactive bias:* There is a risk that the presence of a researcher influences the outcome. This is not perceived as a large risk given a long term collaboration between the company and the university. Furthermore, the main author is also employed at Ericsson and not viewed as an external researcher. However, as the new model was strongly supported by management the interviews are likely to be biased towards the new model to reflect the political drift. In order to reduce this threat, the interviewees were informed that they had been randomly selected. Furthermore, anonymity of the individuals' responses was guaranteed.

- *Correct data interview:* The questions of the interviewer may be misunderstood or the data may be misinterpreted. To avoid this threat, several actions have been taken. First of all, pre-tests were conducted regarding the interviews to ensure a correct interpretation of the questions. Furthermore, all interviews were taped allowing the researcher to listen to the interview again if some parts were misunderstood or unclear.

- *Correct data measurements:* The data sources for requirements waste, faults, and maintenance effort were summarized by the company and the process of data collection and the data sources were not made available to the researchers. In consequence, there is a validity threat regarding potential problems of the rigor of the data collection. In addition, the interpretation of the data is limited due to the high abstraction of the measurements. Hence, the data can only be used as an additional data source for triangulation purposes in order to support the (main) qualitative findings, but not to make inferences such as to which quantified improvement is possible due to the introduction of incremental and agile practices.

Internal Validity

- *Confounding factors influencing measurements:* There is a risk that changes in the performance measurements reported are not solely due to the employment of incremental and agile practices, but also due to confounding factors. As the studied company is a complex organization we were not able to rule out confounding factors as an influence on the measurement outcome. In addition one person involved in reporting the measurements were asked about possible confounding factors, such as major difference in the products, or a change in personnel. The response was that the products compared had similar complexity and that

the products were developed by the same work-force. The person believed that changes in the measurements can, at least partly, be attributed to the migration. However, it is important to point out that the main outcome of the study is the qualitative data from the interviews and that the quantitative data was consulted as an additional data-source to identify whether the quantitative data contradicts the qualitative data, which was not the case.

- *Ability to make inferences about improvements (qualitative data):* Another threat to internal validity is that the instrument and analysis did not capture the change due to the migration to incremental and agile development. However, this threat was reduced by explicitly asking for the situation before and after the migration. In addition, the interviewer asked follow-up questions whenever it was unclear whether an issue was only considered important for one of the process models by the interviewee, or whether the issue was equally relevant to both development approaches. Hence, this threat to validity is considered being under control.

External Validity

- *Process models:* It is impossible to collect data for a general process, i.e., as described in the literature. Both the plan-driven and the incremental and agile model were adaptations of general processes presented in the literature. This is obvious when it comes to the incremental and agile process, but it is the same for the plan-driven model. It is after all a specific instantiation of the generally described plan-driven model. The incremental and agile model is a little more complicated in the mapping to the general process models since it was inspired by two different approaches: incremental development and agile development. To ensure that the findings are not only relevant for these instantiations, care has been taken to carefully describe the context of the study. Furthermore, Table 4.1 illustrates which practices from the general process models have been employed at the company. As the instantiated model and the general process models share practices lessens learned in this study are of relevance for the general process models as well.

- *A specific company:* A potential threat is of course that the actual case study has been conducted within one company. It has been impossible to conduct a similar study at another company. This type of in-depth study requires a lot of effort and that the research is embedded into the organization, which has made it impossible to approach more than one company. To minimize the influence of the study being conducted at one company, the objective is to map the findings from the company specific processes and issues to general processes and higher

level issues. This allows others to learn from the findings and to understand how the results map to another specific context.

Reliability

- *Interpretation of data:* There is always a risk that the outcome of the study is affected by the interpretation of the researcher. To mitigate this threat, the study has been designed so that data is collected from different sources, i.e., to conduct triangulation to ensure the correctness of the findings. Another risk is that the interpretation of the data is not traceable and very much depended on the researcher conducting the analysis. To reduce the risk a workshop was conducted with both authors of the paper and three company representatives being present (see fifth step in the analysis process presented in Section 4.4.5). In the workshop the steps of the researcher were repeated on a number of issues in order to identify potential problems in the analysis steps and interpretations. The practitioners as well as the authors of the paper agreed on the interpretation of the raw data. Hence, the threat to the interpretation of data is considered under control.

Summary

In summary, the case study has been designed according to guidelines and tactics provided in [45]. Measures have been taken whenever possible to mitigate the risks identified in the design. The objective has been to always work in two dimensions: situation specific and general models. The former will in particular be used when continuing the improvement work at the company, where the findings will drive the further improvement work. This is very much the industry view. The latter represents more of an academic view where the intention has been to understand the issues inhibiting process performance in different process models.

4.5 Qualitative Data Analysis

Section 4.4 explains the classification of issues into general and local. In total 64 issues were identified of which 24 were of general nature and the remaining 40 were local problems relating to experiences of individuals or specific subsystems. We focused the detailed qualitative analysis on issues that received high weights in terms of number of responses for each issue. That gave 13 issues for the detailed analysis of which 2 were considered general, 3 were considered very common and 8 were considered common. An overview of the issues is provided in Table 4.6.

Commonly perceived improvements are shown in Table 4.7. The table shows the ID, commonality, process model (either plan-driven=PD or incremental/agile=IA), and a description of the issue. The improvements explain why specific issues were not that important in agile and incremental development anymore. The general issue F01, for example, was mitigated by improvements in requirements engineering (e.g., I02 and I03). A number of improvements on verification were also identified (I04 and I05), which reduced the effect of issue F03 and F04. That is, incremental and agile development enabled early testing and regular feedback to developers. Furthermore, improvement I06 positively influenced the number of documentation which was raised as an important issue (F06). Overall, the tables indicate that the mentioned improvements were in-line with the classification of issues related to process performance. In the following subsections a detailed description of issues and improvements is provided.

4.5.1 General Issues

The most general issues were related to plan-driven development, one being related to the requirements phase and one to the testing phase.

F01: Requirements change and rework: All requirements had to be ready before the next phase starts. That means, when developing a highly complex system the requirements gathering, specification and validation took a very long time. Furthermore, it was hard to estimate the resources needed for a complex system resulting in a too big scope. As interviewees pointed out *"the problem is that the capacity of the project is only that and that means that we need to get all the requirements, discuss them, take them down, look at them and then fit the people and the time frame that we will usually be given. And this negotiation time that was the part that took so long time. It was always and often frustrating."* Another interviewee added that *"there is always a lot of meetings and discussions and goes back and forth and nobody is putting the foot down."* The long lead times of requirements engineering have negative consequences. The market tended to change significantly in the considered domain. In consequence, a high amount of requirements gathered became obsolete or changed drastically which led to wasted effort as the discarded requirements had been negotiated and validated before or requirements had to be reworked. Requirements changes were caused by a lack of customer communication (i.e., the customer was far away from the point of view of the developers or system managers). In addition, misunderstandings were more likely to happen, which result in changed requirements. Regarding the reasons for inflexibility one interviewee added that *"in the old model because its very strict to these tollgates (quality doors) and so on and the requirement handling can be very complex because the process almost requires to have all the requirements clearly defined in the beginning and you should not change them during the way, its not very flexible."*

Table 4.6: Classification of Identified Issues

ID	Classification	Model	Process Area	Description
F01	General	PD	Requirements	Requirements work was wasted as documented and validated requirements had to be discarded or reworked.
F02	General	PD	Verification	Reduction of test coverage due to limited testing time in the end.
F03	Very Common	PD	Verification	Amount of faults found increased with late testing.
F04	Very Common	PD	Verification	Faults found late in the process were hard and expensive to fix.
F05	Very Common	IA	Verification	LSV cycle times may extend lead-time for package deliveries as if a package was not ready or rejected by testing it had to wait for the next cycle.
F06	Common	PD	Requirements	Too much documentation was produced in requirements engineering that was not used in later stages of the process.
F07	Common	PD	Design	Design had free capacity due to long requirements engineering lead times.
F08	Common	PD	Design	Confusion on who implemented which version of the requirements.
F09	Common	PD	Maintenance	High number of corrections for faults reported by customers were released.
F10	Common	PD	Project Mgt.	Specialized competence focus and lack of confidence.
F11	Common	IA	Verification	Low test coverage.
F12	Common	IA	Release	Release was involved too late in the development process.
F13	Common	IA	Project Mgt.	Management overhead due to a high number of teams requiring much coordination and communication.

F02: Reduction of test coverage due to limited testing time in the end: Test coverage in the plan-driven approach was low for multiple reasons. Testing was done late in the project and thus if there were delays before in development testing had to be

compromised as it was one of the last steps in development. As one interviewee put it testing *"takes a long time to get the requirements specification, all the pre-phases, analysis and so on takes a lot of time, design starts too late and also takes a lot of time, and then there is no time for testing in the end"*. Furthermore, too much had to be tested at once after the overall system had been implemented. Due to the complexity of the overall system to verify in the end, testers focused on the same parts of the system twice due to coordination problems instead of covering different parts of the system.

Table 4.7: Commonly Perceived Improvements

ID	Process Area	Description
I01	Requirements	More stable requirements led to less rework.
I02	Requirements	Everything that was started was implemented.
I03	Requirements	Estimations were more precise.
I04	Verification	Early fault detection and feedback from test.
I05	Verification	The lead-time for testing was reduced.
I06	Project Mgt.	Moving people together reduced the amount of documentation that was not reused due to direct communication.

4.5.2 Very Common Issues

Two important issues were identified in the plan-driven development (F03, F04):

F03: Amount of faults found increases with late testing: With late testing one does not know the quality of the system until shortly before release. As testing was not done continuously faults made in the beginning of the implementation were still in the software product. Another issue that increased the number of faults was limited communication between implementation and test. That is, testing started verifying unfinished components of the system which led to a high number of false positives as they did not know the status of the components.

F04: Faults found late in the process were hard and expensive to fix: Late testing resulted in faults hard to fix, which was especially true for faults rooted in the architecture of the system. Changes to the architecture had a major impact on the overall system and required considerable effort. One interviewee reported from experience that *"the risk when you start late is that you find serious problems late in the project phases, and that have occurred a couple of times always causing a lot of problems. Usually the problems I find are not the problems you fix in an afternoon, because they can be deep architectural problems, overall capacity problems and stuff like that which*

is sometimes very hard to fix. So I have always lobbied for having time for a pre-test even if not all the functionality is there."

Issue F05 is related to testing in incremental and agile development:

F05: LSV cycle times may extend lead-time for package deliveries as if a package is not ready or rejected by testing it had to wait for the next cycle. The lead-time of testing was not optimized yet which extended the overall lead time of the development process. An LSV was separated in cycles. Within one cycle (time-window with a fixed end-date) the projects needed to drop their completed component to the LSV. The LSV cycles (4 weeks) did not match with the target dates of the availability of the product on the market. That is, coordination between selling the product and developing the product was complicated. The LSV concept also required a component to wait for another complete LSV cycle if not delivered within the cycle it was supposed to be delivered. Furthermore, if a package was rejected from the LSV due to quality problems and could not be fixed and retested in time, it also had to wait for the next cycle.

4.5.3 Common Issues

The following important issues are related to plan-driven development:

F06: Documentation produced was not used: The interviewees emphasized that quite a high number of documentation was produced in the requirements phase, one interviewee added that *"it (documentation) takes much effort because it is not only that documents should be written, it should be reviewed, then there should be a review protocol and a second round around the table."* One of the reasons mentioned was bad reuse of documentation, which was pointed out by another interviewee saying that *"even though documentation might be good for the quality it might not be good overall because much of the documentation will not be reused or used at all."* Hence, the review of requirements documents required a too high amount of documentation and complex checklists.

F07: Design had free capacity due to long requirements engineering lead times: The requirements lead-time in plan-driven development were quite long. The reasons being that requirements had to be specified in too much detail, decision making took a long time, or requirements resources were tied up because of a too big scope. This had a negative impact on the utilization of personnel. One interviewee nicely summarized the issue saying that *"the whole waterfall principle is not suited for such large projects with so many people involved because half the workforce ends up working for the rest, and I guess thats why the projects were so long. Because you start off with months of requirements handling and during that time you have a number of developers more or less doing nothing."*

F08: Confusion on who implements which version of the requirements: From a design perspective, it was not always clear which version of the requirements should be implemented and by whom. The cause of this problem was that work often started on unfinished or unapproved requirements which had not been properly base-lined.

F09: High number of corrections for faults reported by customers were released: Support was required to release a high number of corrections on already released software. This was due to the overall length of the plan-driven projects resulting in very long release cycles. In consequence, the customers could not wait for the corrections to be fixed for the next release, making corrections a time-pressing issue.

F10: Specialized competence focus and lack of confidence: The competence focus of people in plan-driven development was narrowed, but specialized. This was due to that people were clearly separated in their phases and disciplines, and that knowledge was not well spread among them. Interesting was that not only the specific competence focus was recognized as an issue, but also the focus on confidence. One interviewee described the relevance of confidence by saying *"It is not only competence, it is also confidence. Because you can be very competent, but you are not confident you will not put your finger down and say this is the way we are going to do it, you might say it could be done in this way, or in this way, or also in these two ways. This will not create a productive way of working. Competence is one thing, confidence is the other one required."*

Important issues in incremental and agile development are:

F11: Low test coverage: The reasons for low test coverage changed in incremental/agile development and were mainly related to the LSV concept. Quality testing takes too much time on the LSV level, the reason being that there was a lack of powerful hardware available to developers to do quality testing earlier. In consequence, there was a higher risk of finding faults late. Furthermore, the interviewees had worries on the length of the projects as it would be hard to squeeze everything into a three month project (including developing configurations and business logic, testing etc.). In addition to that test coverage was influenced negatively by a lack of independent verification and validation. That is, developers and testers in one team were influencing each other what to test. In consequence, the testing scope was reduced.

F12: Release personnel was involved too late in the development process: This means that release personnel got the information required for packaging the product after requirements, implementation and testing were finished. In consequence, the scope of the product was not known to release and came as a surprise. With regard to this observation one interviewee stated that *"In release we are supposed to combine everything and send it to the market, we were never involved in the beginning. We can have problems with delivering everything that we could have foreseen if we were involved early."* Furthermore, the requirements were not written from a sales perspective, but

mainly from a technical perspective. This made it harder for release to create a product that is appealing to the customer. During the interviews it was explicitly mentioned that this situation has not changed with the introduction of an incremental and agile development approach.

F13: Management overhead due to a high number of teams requiring much coordination and communication: Many small projects working toward the same goal required much coordination and management effort. This included planning of the technical structure and matching it against a time-line for project planning. Thus, project managers had much more responsibility in incremental and agile development. Furthermore, there was one more level of management (more team leaders) required for the coordination of the small teams. The interviewees also had worries that the added level of management had problems to agree on overall product behavior (hardware, application, performance, overall capacity) which delayed decision making. Thus, decisions were not taken when they were needed.

4.5.4 Comparison of Issues

Table 4.6 clearly shows that a majority of general problems was related to plan-driven development. Furthermore, only one issue raised for the incremental and agile model was considered very common (none was general), while four issues for the plan-driven approach were considered general or very common. It is hence clear that the change to an incremental and more agile development model was perceived as having addressed some of the main issues raised for the plan-driven approach. Having said this, it does not mean that the new development approach is unproblematic. However, the problems are at least not perceived as commonly as for plan-driven development. Furthermore, the problem related to test coverage was still perceived as present, but the severity and nature of the issue has changed for the better. Additional detail on improvements and open issues based on the comparison is provided in Section 4.7.

4.5.5 Commonly Perceived Improvements

The following improvements due to the introduction of incremental and agile development practices were mentioned by the interviewees:

I01: More stable requirements led to less rework and changes: Requirements were more stable as requirements coming into the project could be designed fast due to that they were implemented in small coherent packages and projects. That is, the time window was much smaller and the requirements were thus not subjected to change to the same degree. Furthermore, the flexibility was higher in terms of how to specify the requirements. For example, requirements with very low priorities did not need to be

specified in detail. If requirements are just seen as the whole scope of the system, this distinction is not made (as is the case in plan-driven development). Also, the communication and interaction between design and requirements improved, allowing clarifying things and thus implementing them correctly. This communication was improved, but not to the degree as the communication between design and implementation had been improved (see issues for test/ design). One interviewee summarized the increased flexibility by saying that *"within the new ways of working its easier to steer around changes and problems if you notice something is wrong, its much easier to change the scope and if you have change requests on a requirement, I think its more easy."*

I02: Everything that is started is implemented: If a requirement was prioritized it was implemented, the time of implementation depending on the position of the requirement in the priority list. As one interviewee (requirements engineer) reported, before a large part of all requirements engineering work was waste, while now only approximately 10 % of the work is wasted. In partuclar, the new situation allows to complete tasks continuously with not being so dependend on others to be ready, which was explained by one interviewee saying that *"when you talk about the waterfall you always end up in a situation where everybody had to be ready before you continue with next task, but with the new method what we see is that one activity is done, they can pick the next to do, they are not supposed to do anything else."*

I03: Estimations are more precise: The effort can be estimated in a better way as there were less requirements coming into the project, and the requirements were more specific. Furthermore, the incremental and agile model contributed to more realistic estimations. With the plan-driven approach the deadlines and effort estimates were unrealistic when being compared to the requirements scope. When only estimating a part of the prioritized list (highest priority requirements first), then the estimations became much more realistic.

I04: Early fault detection and feedback from test: Problems could be traced and identified much easier as one component was rejected back to the project if a problem occurs. The ability to reject an increment back to a development team has advantages as pointed out by an interviewee stating the following: *"I know that it will be tougher for the design units to deliver the software to testing in incremental and agile development than it was in waterfall because if they (design and implementation) don't have the proper quality it (the increment) will be rejected back to the design organization. This is good as it will put more pressure on the design organization. It will be more visible, you can always say it does not work, we can not take that. It will be more visible to people outside (management)."* Besides that, there was better focus on parts of the system and feedback was provided much earlier. Furthermore, the understanding of testing priorities was improved due to the explicit prioritization of features in requirements engineering. These benefits were summarized by an interviewee who pointed out that

"testing is done on smaller areas, providing better focus. Everything will improve because of the improved focus on feature level and the improved focus of being able to come through an LSV cycle. We will catch the need for rework earlier. The feedback loop will be shorter." With that the interviewee already points to the improvement in lead-time (I05).

I05: The lead-time for testing is reduced: Time of testers was used more efficiently as in small teams, it was easier to oversee who does what. That is, different people in a team did not do the same things twice anymore. Furthermore, parallelization was possible as designers were located close to testers who could do instant testing when some part of the subsystem had been finished.

I06: Moving people together reduced the amount of documentation: People worked in cross-functional and small teams. As the teams were cross-functional less documentation was required as it was replaced with direct communication. That is, no handover items were required anymore as input from previous phases because people were more involved in several phases now. The perceived improvement with regard to communication was pointed out by one interviewee saying that *"now we are working in small teams with 6 people, something like that. It is pretty much easier to communicate, we have these daily meetings. Each one knows what the other one days just this day. The next day we have a follow up meeting, this was done yesterday and I will proceed it today. Might take a while to have those meetings because you have it each day, but it is 15 minutes that is still very useful."*. Another interviewee talked about walls being broken down between the design/implementation and testing organization saying *"We have a better way of working between test and design and they are working side by side so to say. We could do it even better and we work side by side and take small steps. We look at what we test, we look at what part we could start function test on and then we implement it. This wall is totally broken now between our test and design organization."*

4.6 Quantitative Data Analysis

An overview of the quantitative data is used to confirm or contradict the findings of the qualitative analysis. This section just presents the data, its implications together with the qualitative results are discussed in Section 4.7.

4.6.1 Requirements Waste

Requirements are considered waste if they have been elicited, documented and reviewed, but are not implemented. The absolute number and ratio of the number of

requirements that were implemented and discarded are shown in Figure 4.3. The data includes two products as well as two generations of an additional product developed at the studied development site.

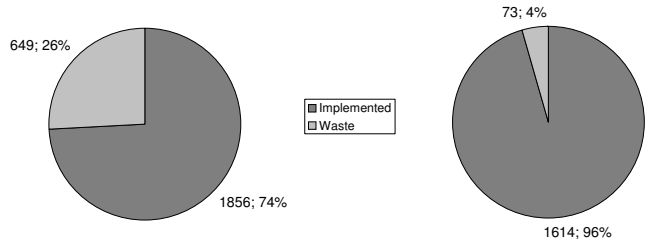


Figure 4.3: Requirements Waste - Plan-Driven (left) vs. Incremental and Agile (right)

Furthermore, the number of change requests per requirement decreased for the same products. Change requests require adjustments and extensions to the requirements. After introducing incremental and agile practices, the number of change requests per requirement decreased from 0.076 to 0.043. Thus, the requirements became more stable.

4.6.2 Software Quality

Table 4.8 shows the fault-slip-through before and after the introduction of agile and incremental development. The system testing phase of plan-driven development is comparable to the test on the LSV level in the new development approach. As mentioned earlier, the fault-slip shows how many faults have been discovered in a specific phase that should have been found earlier. In this case, in total 30 faults should have been found before system test, and 20 faults should have been found in before LSV testing. Comparing this with the overall amount of faults considered, then 31 % of faults slipped through earlier testing phases in plan-driven development, and only 19 % in the new development model.

Table 4.8: Fault Slip Before System Test / LSV

Test	Number of Faults	Slippage
System Test (Plan-Driven)	30	31 %
LSV (Incremental and Agile)	20	19 %

Therefore, the data is an indication that the testing efficiency of functional testing of the packages before delivered to the LSV and basic unit testing by programmers has been improved.

Figure 4.4 shows the maintenance effort for products on the market. The maintenance effort includes costs related to fixing faults that have been found and reported by the customers. Thus, those faults should have been found earlier in testing. The figure shows that the maintenance costs were constantly increasing when new products were released on the market. Projecting the increase of costs in previous years into the future (dashed line showing the maintenance cost baseline) the costs would be 40 % higher than in the year 2005.

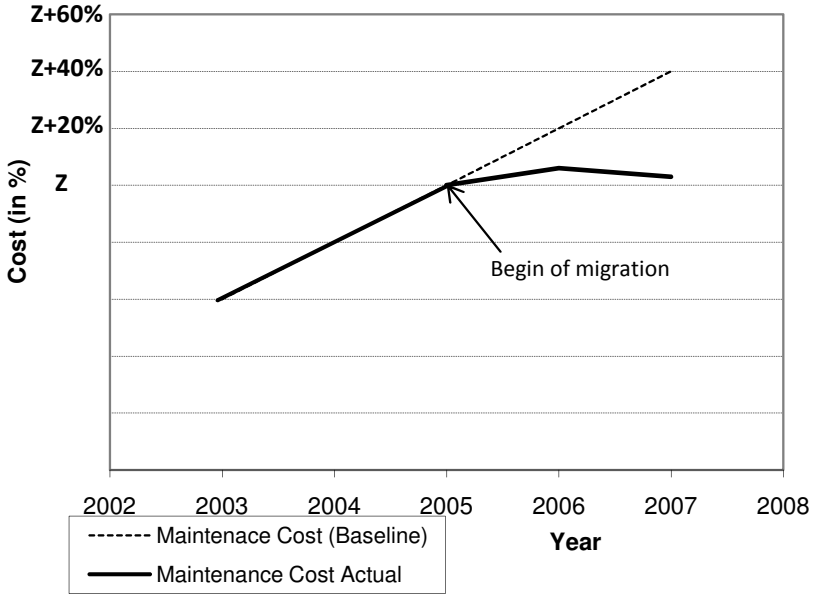


Figure 4.4: Maintenance Effort

After introducing incremental and agile practices the actual cost (bold line) still increased, but the slope of the curve was much smaller. In 2006, after the introduction of incremental and agile practices has been further progressed a slight decrease in maintenance cost is visible. Thus, this is an indication of improved quality assurance which was visible in the fault-slip-through, but is also an indication for improvements in the actual system testing.

4.7 Discussion

The discussion draws together the results from the qualitative and quantitative analysis. It is divided in two parts, namely improvement areas and open issues. Open issues are problems that still have to be addressed after moving from plan-driven to incremental and agile development.

4.7.1 Improvement Areas

Release frequency: The qualitative data showed that a higher release frequency is possible due to building the product in increments using the LSV concept. However, there is no conclusive evidence that the overall productivity of the development has increased. That is, the same workforce does not produce the same amount of software artifacts in shorter time than before. Instead, the company is able to deliver functionality more frequently which benefits the organization. Frequent releases lead to earlier return on investments. In plan-driven development, a large up-front investment is required which starts paying off when the overall development has been completed.

Reduction in waste: A clear improvement can be seen in the reduction of waste, shown in the qualitative analysis which is supported by the quantitative analysis. Furthermore, the number of change requests have been reduced which is an indicator for that the requirements are a better reflection of the customers' needs than with the plan-driven model. The benefits of this are also explicitly mentioned in the qualitative data (see I01 in Section 4.5). Overall, improvements related to waste in requirements can be considered essential as this type of waste has been identified as one of the most crucial problems in plan-driven development (see F01 in Section 4.5). A good reflection of the needs of the users in the requirements is also essential to make sense of the improvement that everything that is started is implemented (see I03 in Section 4.5). If the requirements would not be reflected in the current needs, this implementation could be considered waste, even though it has been identified as an improvement. Finally, the reduced scope in the incremental and agile model helps to have more accurate estimations (I06), meaning that the requirements scope is set appropriately for each increment. Thus, it is less likely that requirements have to be discarded due to inaccurate planning.

Software Quality Improvements: The quantitative data shows improvement in early testing done before system testing (LSV), reflected in a reduced fault-slip-through in comparison to the plan-driven approach. Furthermore, the constantly rising maintenance effort decreased after introducing incremental and agile practices, even though there have not been any major tendencies for it to go below the level of 2005 (see Figure 4.4). In the qualitative data, we identified one improvement raised in the interviews.

That is, testing has improved due to early fault detection and feedback from test (see I03 in Section 4.5). Furthermore, if an increment is dropped to the LSV for test one can trace which increments are of high or low quality and who is responsible for them. Consequently, this creates incentives for teams to deliver high quality as their work result is visibly linked to them. By testing early many verification issues identified in plan-driven development can be addressed. These are reduction of test coverage due to complex testing in the end (F02), increase of the number of faults discovered with late testing (F03), and that problems are harder to fix when discovered late (F02). The study shows that even though there has been improvements in testing, very important and important issues relate to verification in incremental and agile developed, further discussed in the context of open issues.

Improved Communication: The qualitative data suggests an improvement in communication when moving people together (I06). This positively affects several issues that have been identified for plan-driven development. Firstly, the amount of documentation can be reduced because much of the documentation was related to hand-overs between phases (F06). As a project team focuses on several phases now, direct communication can replace parts of the documentation. Furthermore, in plan-driven development the knowledge of people is very specialized and they is a lack of confidence. This can be hindering in the beginning when moving from plan-driven to incremental and agile practices as having small teams requires very broad knowledge of the team members (see for example [29]). However, at the same time face-to-face interaction helps team members to learn from each other and gain insight and understanding of the overall development process [39].

The perceived improvements are further strengthened by the fact that incremental and agile practices have not been employed for a long time at the studied companies. The positive results already achieved are also important as a motivator and buy-in to further progress with the agile implementation by adding further agile practices such as test driven development or pair programming.

4.7.2 Open Issues

Based on the classification, the most important issues that remain when moving to incremental and agile development are related to verification, project management, and release planning.

Verification: For verification, the improvement of reduced lead-times for testing have been identified (I06). However, the issue relates to that the LSV cycle times are not optimized and that there is room of improvement to shorten the lead-time of testing, the issue being the only issue related to incremental and agile development classified as very important. Thus, although the lead time is perceived to have improved, it is still an

area needing attention. Furthermore, the test coverage is considered a problem in both development models (see F02 for plan-driven development and F11 in incremental and agile development), even though the classification shows that it is less common for the latter development model. The descriptions of the issues related to test coverage show that test coverage is a problem due to different reasons in both development models. In plan-driven development, the test coverage is reduced because too much has to be tested at once, and the testing time is always compromised in the end of the plan-driven project. In incremental and agile development though the problems are more specific: quality testing in the LSV takes too much time; the project cycles are too short to squeeze everything into the project; and there is a lack of independent verification for basic and component testing. This still being a problem, it is less common in incremental and agile than in plan-driven development. However, due to the explicitly identified problems in testing it is clear that there is room for improvement to achieve more significant improvements, e.g., by implementing test-driven development or increase the degree of automated testing to speed up the testing process.

Management Overhead: Due to the high number of teams, the work on the team level gets more clear and simplistic with an incremental and agile model. However, many projects working toward the same goal have to be coordinated. As discussed earlier (see F13 in Section 4.5) this requires much communication and planning involving many different people. Therefore, this issue is specifically related to the scalability of incremental and agile methods. To address the issue, the first step taken was the anatomy plan which helps to structure the system and dependencies. This is used as input for deciding on the order of projects to build the increments.

Release Project: The release project is responsible for bringing the product into a shippable state. The release project is very specific for the company as it is related to building customizable solutions. That is, in the release project a tool has to be created that allows the selection of features so that the product is customizable. As raised in the earlier discussion (F12) people of the release project are involved too late in the development process and thus the product is not viewed from a commercial perspective. Consequently, an action for improvement would be to integrate people from the release project already in the requirements engineering phase.

4.7.3 Implications

The results of the case study indicate that it is beneficial for large-scale organization to move from plan-driven to incremental and agile development. In fact, the study came to the surprising result that plan-driven development is not suitable for a large-scale organization as it produces too much waste in development (specifically in the requirements phase). The most pressing issues identified in the organization were in-

fact related to the plan-driven approach. On the other hand, important improvements can be achieved by moving from plan-driven to incremental and agile development. We have shown that specific areas of improvements are reduction of waste and better responsiveness to market changes. Furthermore, there are opportunities for faster return of investment. However, the case study also shows that even though benefits can be gained on the one hand, challenges are raised on the other hand. Areas that specifically show room for improvements are testing and support for coordinating a large number of development teams. The challenges are important to recognize and address when further progressing with the agile implementation.

4.8 Conclusions and Future Work

This paper investigates the effect of migrating from a plan-driven to an incremental and agile development approach. The study shows that the most commonly perceived problems in the development models can be found in plan-driven development, and moving from plan-driven to agile and incremental development allows to improve on these most common issues. Returning to the research questions and propositions, we can conclude:

Issues: Several issues were identified for both the plan-driven and the incremental and agile approach. However, more commonly perceived issues across roles and systems were identified for the plan-driven approach.

General issues: The two most commonly perceived issues overall were identified for the plan-driven approach: 1) requirements change and rework; and 2) reduction of test coverage due to limited test time at the end. Several other issues were identified both for the plan-driven and the incremental and agile approach.

Performance measures: It was only possible to collect two comparable performance measures: waste in terms of investment in requirements never delivered and fault-slip-through. Both these measures were in favor of the incremental and agile model.

Proposition 1 is partially true, i.e. both different and in some cases similar issues were identified for the two models. Proposition 2 holds. Improvements in the quantitative data have been observed and thereby supporting the primary evidence reported for the qualitative data.

Thus, in summary the main improvements identified are 1) ability to increase release frequency and shorten requirements lead-times; 2) significant reduction of waste and better reflection of the current customers' needs measured as reduced number of change requests; 3) improvements in software quality for basic testing (unit and component testing) and overall system quality, and 4) improved communication which facili-

tates better understanding and allows to reduce documentation. However, incremental and agile methods raise a number of challenges at the same time, which are: 1) needs for coordinating testing and increase test coverage; 2) support for coordinating a high number of teams and making decisions related to planning time-lines for concurrent projects; and 3) integration of release projects in the overall development process. In future work, more qualitative as well as quantitative studies are needed to compare development models for large-scale development.

4.9 References

[1] David J. Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results (The Coad Series)*. Prentice Hall PTR, 2003.

[2] Bouchaib Bahli and El-Sayed Abou-Zeid. The role of knowledge creation in adopting xp programming model: An empirical study. In *ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society*, 2005.

[3] Richard Baskerville, Balasubramaniam Ramesh, Linda Levine, Jan Pries-Heje, and Sandra Slaughter. Is internet-speed software development different? *IEEE Software*, 20(6):70–77, 2003.

[4] Kent Beck. Embracing change with extreme programming. *IEEE Computer*, 32(10):70–77, 1999.

[5] Oddur Benediktsson, Deee Dalcher, and Haaa thorbergsson. Comparison of software development life cycles: a multiproject experiment. *IEE Proceedings Software*, 153(3):323–332, 2006.

[6] Martina Ceschi, Alberto Sillitti, Giancarlo Succi, and Stefano De Panfilis. Project management in plan-based and agile companies. *IEEE Software*, 22(3):21–27, 2005.

[7] David Cohen, Mikael Lindvall, and Patricia Costa. *Advances in Computers, Advances in Software Engineering*, chapter An Introduction to Agile Methods. Elsevier, Amsterdam, 2004.

[8] David Cohen, Gary Larson, and Bill Ware. Improving software investments through requirements validation. In *Proceedings of the 26th Annual NASA God-*

dard Software Engineering Workshop (SEW 2001), page 106, Washington, DC, USA, 2001. IEEE Computer Society.

[9] Aldo Dagnino, Karen Smiley, Hema Srikanth, Annie I. Antón, and Laurie A. Williams. Experiences in applying agile software development practices in new product development. In *Proceedings of the IASTED Conference on Software Engineering and Applications (IASTED-SEA 2004*, pages 501–506, 2004.

[10] Liangtie Dai and Wanwu Guo. Concurrent subsystem-component development model (cscdm) for developing adaptive e-commerce systems. In *Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2007)*, pages 81–91, 2007.

[11] Lars-Ola Damm and Lars Lundberg. Company-wide implementation of metrics for early software fault detection. In *Proceedings of the 9th International Conference on Software Engineering (ICSE 2007)*, pages 560–570, 2007.

[12] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006.

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

[14] Richard E. Fairley and Mary Jane Willshire. Iterative rework: The good, the bad, and the ugly. *IEEE Computer*, 38(9):34–41, 2005.

[15] Geir Kjetil Hanssen, Hans Westerheim, and Finn Olav Bjørnson. Using rational unified process in an sme - a case study. In *Proceedings of the 12th European Conference on Software Process Improvement (EuroSPI 2005)*, pages 142–150, 2005.

[16] Werner Heijstek and Michel R. V. Chaudron. Evaluating rup software development processes through visualization of effort distribution. In *Proceedings of the 34th Conference on Software Engineering and Advanced Applications (SEAA 2008)*, pages 266–273, 2008.

[17] Michael Hirsch. Moving from a plan driven culture to agile development. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, page 38, 2005.

[18] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an agile methodology implementation. In *Proceedings of the 30th EUROMICRO Conference (EUROMICRO 2004)*, pages 326–333, 2004.

[19] J Jarzombek. The 5th annual jaws s3 proceedings, 1999.

[20] Jim Johnson. Keynote speech: Build only the features you need. In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002)*, 2002.

[21] Caspers Jones. *Patterns of Software Systems: Failure and Success*. International Thomson Computer Press, 1995.

[22] Daniel Karlström and Per Runeson. Combining agile methods with stage-gate project management. *IEEE Software*, 22(3):43–49, 2005.

[23] Alan S. Koch. *Agile software development: evaluating the methods for your organization*. Artech House, Boston, 2005.

[24] Phillip A. Laplante and Colin J. Neill. Opinion: The demise of the waterfall model is imminent. *ACM Queue*, 1(10):10–15, 2004.

[25] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Pearson Education, 2003.

[26] Katiuscia Mannaro, Marco Melis, and Michele Marchesi. Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies. In *Proceedings of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, pages 166–174, 2004.

[27] Angela Martin, Robert Biddle, and James Noble. The xp customer role in practice: Three studies. In *Agile Development Conference*, pages 42–54, 2004.

[28] Pete McBreen. *Questioning Extreme Programming*. Pearson Education, Boston, MA, USA, 2003.

[29] H. Merisalo-Rantanen, Tuure Tuunanen, and Matti Rossi. Is extreme programming just old wine in new bottles: A comparison of two cases. *J. Database Manag.*, 16(4):41–61, 2005.

[30] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software, in print*, 82(9):1479–1490, 2009.

[31] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, pages 401–404, 2009.

[32] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *Proceedings of the 10th International Conference on Product Focused Software Development and Process Improvement (PROFES 2009)*, pages 386–400, 2009.

[33] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit (The Agile Software Development Series)*. Addison-Wesley Professional, 2003.

[34] Lbs Raccoon. Fifty years of progress in software engineering. *SIGSOFT Softw. Eng. Notes*, 22(1):88–104, 1997.

[35] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[36] Ken Schwaber. *Agile project management with Scrum*. Microsoft Press, Redmond, Wash., 2004.

[37] Alberto Sillitti, Martina Ceschi, Barbara Russo, and Giancarlo Succi. Managing uncertainty in requirements: A survey in documentation-driven and agile companies. In *Proceedings of the 11th IEEE International Symposium on Software Metrics (METRICS 2005)*, page 17, 2005.

[38] Matt Stephens and Doug Rosenberg. *Extreme Programming Refactored: The Case Against XP*. Apress, Berkeley, CA, 2003.

[39] Harald Svensson and Martin Höst. Introducing an agile process in a software maintenance and evolution organization. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, pages 256–264, 2005.

[40] Bjørnar Tessem. Experiences in learning xp practices: A qualitative study. In *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004)*, pages 131–137, 2003.

[41] Michael Thomas. It projects sink or swim. British Computer Society Review 2001, 2001.

[42] Piotr Tomaszewski. *Software development productivity - evaluation and improvement for large industrial projects*. PhD thesis, Detp. of Systems and Software Engineering, Blekinge Institute of Technology, 2006.

[43] Andrew Wils, Stefan Van Baelen, Tom Holvoet, and Karel De Vlaminck. Agility in the avionics software world. In *XP*, pages 123–132, 2006.

[44] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in Software Engineering: An Introduction (International Series in Software Engineering)*. Springer, 2000.

[45] Robert K. Yin. *Case Study Research: Design and Methods, 3rd Edition, Applied Social Research Methods Series, Vol. 5*. Prentice Hall, 2002.

Chapter 5

A Comparison of Issues and Advantages in Agile and Incremental Development between State of the Art and an Industrial Case

Kai Petersen and Claes Wohlin
Published in Journal of Systems and Software

5.1 Introduction

The nature of software development has changed in recent years. Software is now included in a vast amount of products (cars, entertainment, mobile phones) and is a major factor determining whether a product succeeds. In consequence it becomes more and more important to be flexible in handling changing requirements to meet current customer needs and to be able to deliver quickly to the market. As a solution, agile methods have started to be adopted by industry and recent studies have been focusing on evaluating agile and incremental development models.

A systematic review [4] identified and analyzed studies on agile software development. Thirty three relevant studies were identified of which 25 investigated Extreme Programming (XP). Furthermore, only three papers investigated projects with more than 50 people involved in total. Thus, the results so far are hard to generalize due to the focus on one specific method and small projects. In order to address this research gap a large-scale implementation of a set of agile and incremental practices is investigated through an industrial case study at Ericsson AB. In particular, issues and advantages of using agile and incremental methods are extracted from existing studies and are compared with the results of the case study. Three subsystems have been investigated at Ericsson through 33 interviews, covering persons from each subsystems and different roles.

The incremental and agile model used at the company is a selection of agile and incremental practices, so it cannot be mapped one to one to the models presented in literature. However, the company's agile and incremental model uses practices from SCRUM (SC), XP, and incremental and iterative development (IID). Some of the key practices used at the company are, for example, the devision of internal and external releases, small and motivated teams developing software in three month projects (timeboxing), frequent integration of software, and always developing the highest prioritized features first.

The main contribution of the study is to help in the decision of adopting agile methods and showing the problems that have to be addressed as well as the merits that can be gained by agile methods. Based on this, the following objectives are formulated for the study:

- Illustrate one way of implementing incremental and agile practices in a large-scale organization.
- Provide an in-depth understanding of the merits and issues related to agile development.
- Increase the generalizability of existing findings by investigating a different study context (large scale and telecommunication domain) and comparing it to state of the art.

The structure of the chapter is shown in Figure 5.1: Section 5.2 presents the state of the art, summarizing issues and advantages identified in literature. Thereafter, the investigated process model is described in Section 5.3, and the practices applied in the company's model are mapped to the practices associated with incremental and agile development, XP and SCRUM for the purpose of generalizability. Section 5.4 illustrates the research method, which constitutes the context in which the study is conducted, the

data collection procedures, and a description of the data analysis methods. The results of collected data (Section 5.5) show issues and advantages identified in the case study, as well as a mapping between the findings of the case study and the state of the art described in Section 5.2 (see 5.1). Sections 5.6 and 5.7 present the implications of the results. The implications of the comparison of state of the art and case are discussed, and the mapping (5.3) can be used to discuss generalizability and comparability of results.

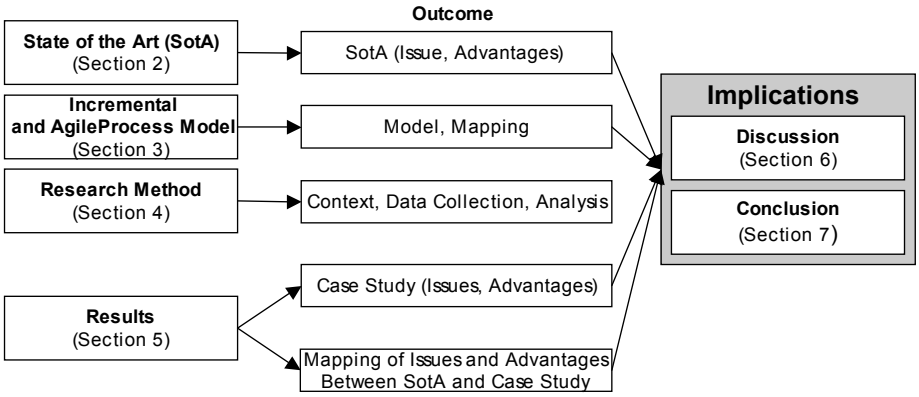


Figure 5.1: Structure of the Chapter

5.2 State of the Art

The studies presenting advantages and issues of agile and incremental methods have been identified in a recent systematic review of empirical studies in agile software development [4]. The issues and advantages presented here are from the review as well as looking at the original articles included in the review. Furthermore, one further article not included in the systematic review has been identified ([15]). The issues and advantages are presented as tables receiving an ID (A01-A11, I01-I10) which is used as a reference when comparing the issues and advantages identified in this case study with the findings in state of the art. The tables also contain information of which development models are related to a specific issue and how large the project was (measured in number of team members).

Table 5.1 provides an overview of advantages of agile and incremental development that have been empirically shown in the studies identified in [4]. The advantages that

have been shown for agile methods are clearly dominated by studies that investigated XP or a modified version of XP ([6]). Two advantages (A01, A02) have been shown for SCRUM as well. The size of the projects is quite small (up to 23) and for many projects the size has not been reported. The main advantages are related to benefits of communication leading to better learning and knowledge transfer (A01, A03, A07). Furthermore, it is emphasized that people feel comfortable using agile methods (A08, A10). Also, customers appreciate agile methods as they provide them with the opportunity of influencing the software process and getting feedback (A09). The same is true vice versa, meaning developers also appreciate the presence of customers (A02). Developers value the technical focus of agile methods increasing their motivation (A05). There is also a perception of increased quality in software products (A11) and higher productivity (A10) when using pair programming.

Besides the advantages, agile and incremental models face a number of issues that are summarized in Table 5.2. Studies identifying issues reveal the same pattern as studies identifying advantages, that is small projects have been studied and the main focus has been on XP. The positive effects that have been shown for pair programming (A03, A10, A11), like higher quality and productivity and facilitated learning, need to be seen alongside a number of issues. That is, pair programming is perceived as exhausting (I04) and requires partners with equal qualifications (I10). Agile can also be considered an exhausting activity from customers' point of view as the customer has to commit and be present throughout the whole development process. Team related issues are that members of teams have to be highly qualified and inter-team communication suffers (I05, I10). From a management point of view two issues are identified, namely that they feel threatened by the empowerment of engineers (I07) and that technical issues are raised too early (I08). Furthermore, agile projects do not scale well (I03) and have too little focus on architecture development (I01). Agile also faces implementation problems when realizing continuous testing as this requires much effort (I01).

The advantages and issues of the state of the art shown in Tables 5.1 and 5.2 are used as an input for the comparison with the results from the case study in Section 5.5.

5.3 Incremental and Agile Process Model

The process model used at the company is described and thereafter its principles are mapped to incremental and iterative development, SCRUM, and XP. The model is primarily described to set the context for the case study, but the description also illustrates how a company has implemented an incremental and agile way of working.

Table 5.1: Advantages in Incremental Agile Development (State of the Art)

ID	Advantages	Model	Size	Study
A01	Better knowledge transfer due to better communication and frequent feedback from each iteration.	XP/XP/XP/SC&XP	9/-/-/7	[1, 6, 18, 15]
A02	Customers are perceived by programmers as very valuable allowing developers to have discussions and get early feedback.	XP/SC/XP/XP	-/6/-/6	[6, 9, 18, 19]
A03	Pair programming facilitates learning if partners are exchanged regularly.	XP	6	[19]
A04	Process control, transparency, and quality are increased through continuous integration and small manageable tasks.	XP	-	[6]
A05	XP is very much technical-driven empowering engineers and thus increases their motivation.	XP	-	[6]
A07	Small teams and frequent face-to-face meetings (like planning game) improves cooperation and helps getting better insights in the development process.	XP(modification)	-	[18]
A08	The social job environment is perceived as peaceful, trustful, responsible, and preserving quality of working life.	XP	23	[16]
A09	Customers appreciate active participation in projects as it allows them to control the project and development process and they are kept up to date.	XP	4	[5]
A10	Developers perceive the job environment as comfortable and they feel like working more productive using pair programming.	XP	-	[10]
A11	Student programmers perceive the quality of code higher using pair programming	XP	-	[10]

Table 5.2: Issues in Incremental and Agile Development (State of the Art)

ID	Issues	Model	Size	Study
I01	Realizing continuous testing requires much effort as creating an integrated test environment is hard for different platforms and system dependencies.	XP(modification)	-	[18]
I02	Architectural design does not have enough focus in agile development leading to bad design decisions.	gen./gen.	-/-	[12, 17]
I03	Agile development does not scale well.	gen.	-	[3]
I04	Pair programming is perceived as exhaustive and inefficient.	XP/XP/XP	4/12/6	[5, 8, 19]
I05	Team members need to be highly qualified to succeed using agile.	XP	6	[14]
I06	Teams are highly coherent which means that the communication within the team works well, but inter-team communication suffers.	XP/SC&XP	-/7	[6, 15]
I07	The empowerment of engineers makes managers afraid initially, and thus requires sufficient training of managers.	XP	-	[6]
I08	Implementation starts very early, thus technical issues are raised too early from a management point of view.	XP	-	[6]
I09	On-site customers have to commit for the whole development process which puts them under stress.	XP	16	[11]
I10	From the perspective of students, pair programming is not applicable if one partner is much more experienced than the other.	XP	-	[13]

5.3.1 Model Description

Due to the introduction of incremental and agile development at the company the following company specific practices have been introduced:

- *Small Teams:* The first principle is to have small teams conducting short projects lasting three months. The duration of the project determines the number of requirements selected for a requirement package. Each project includes all phases of development, from pre-study to testing. The result of one development project

is an increment of the system and projects can be run in parallel.

- *Implementing highest Priority Requirements:* The packaging of requirements for projects is driven by requirement priorities. Requirements with the highest priorities are selected and packaged to be implemented. Another criterion for the selection of requirements is that they fit well together and thus can be implemented in one coherent project.

- *Use of Latest System Version:* If a project is integrated with the previous baseline of the system, a new baseline is created. This is referred to as the latest system version (LSV). Therefore, only one product exists at one point in time, helping to reduce the effort for product maintenance. The LSV can also be considered as a container where the increments developed by the projects (including software and documentation) are put together. On the project level, the goal is to focus on the development of the requirements while the LSV sees the overall system where the results of the projects are integrated. When the LSV phase is completed, the system is ready to be shipped.

- *Anatomy Plan:* The anatomy plan determines the content of each LSV and the point in time when a LSV is supposed to be completed. It is based on the dependencies between parts of the system developed which are developed in small projects, thus influencing the time-line in which projects have to be executed.

- *Decoupling Development from Customer Release:* If every release is pushed onto the market, there are too many releases used by customers that need to be supported. In order to avoid this, not every LSV has to be released, but it has to be of sufficient quality to be possible to release to customers. LSVs not released to the customers are referred to as potential releases. The release project in itself is responsible for making the product commercially available and to package it in the way that the system should be released.

In Figure 5.2 an overview of development process is provided. At the top of Figure 5.2 the requirements packages are created from high priority requirements stored in the repository. These requirements packages are implemented in projects (for example Project A-N) resulting in a new increment of the product. Such a project has a duration of approximately three months (time-boxed). When a project is finished developing the increment, the increment is integrated with the latest version of the system, referred to as last system version (LSV). The LSV has a pre-defined cycle (for example, projects have to drop their components within a specific time frame to the LSV). At the bottom of the Figure, the different releases of the system are shown. These are either potential releases or customer releases.

The development on project level is run as increments, i.e. each project delivers one increment to the LSV. Within each project, the development is done using iterations and some of the practices from agile development. The process flow works as follows: A set of requirements comes into a new project having a duration of three months. This set of requirements should lead to a new increment. The project is run as a number of iterations. An iteration takes approximately two weeks. Each iteration is a continuous flow with the following steps:

1. Requirements are designed and implemented.

2. The implementation is deployed within the test environment (including test cases).

3. The results from the executed test are monitored.

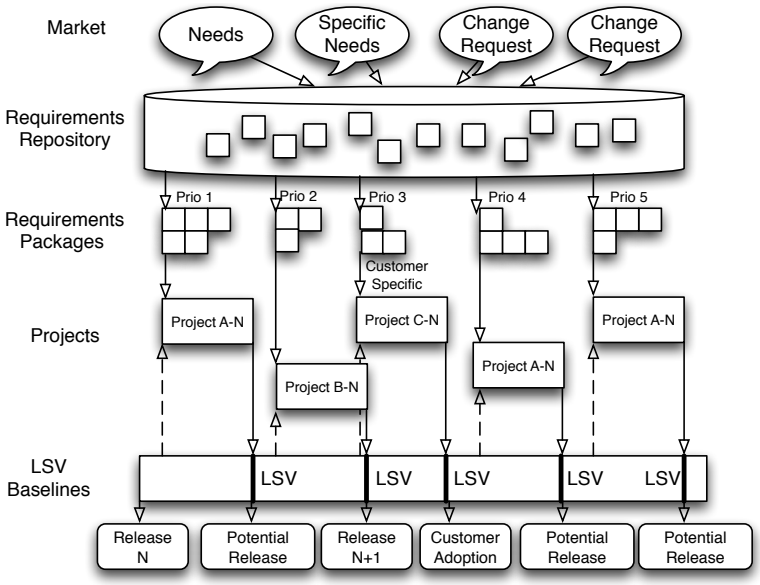


Figure 5.2: Development Process

5.3.2 Mapping

The principles used in the process model at the company (C) are mapped to the ones in incremental and iterative development (IID), extreme programming (XP), and SCRUM (SC). In Table 5.3 we show which principles used in incremental and iterative development (IID), extreme programming (XP), and SCRUM (SC) are used in the process model at the company (C). The Table (created based on the information provided in [7]) shows that 4 out of 5 incremental principles are fulfilled. Furthermore, the model used at the company shares 7 out of 13 principles with XP and 6 out of 11 principles with SCRUM. If many practices are fulfilled in the case study (which is the case for IID and the agile models) we argue that lessens learned provide valuable knowledge of what happens when the process models are transferred to industry in a given context.

Table 5.3: Mapping

Principle	IID	XP	SC	C
Iterations and Increments	√	√	√	√
Internal and External Releases	√			√
Time Boxing	√	√	√	√
No Change of Started Projects	√		√	√
Incremental Deliveries	√			√
On-site Customer		√	√	
Frequent Face-to-Face Interaction		√	√	√
Self-organizing Teams		√	√	
Empirical Process		√	√	
Sustainable Discipline		√		
Adaptive Planning		√	√	
Requirements Prioritization		√	√	√
Fast Decision Making			√	
Frequent Integration		√	√	√
Simplicity of Design		√		
Refactoring		√		
Team Code Ownership		√		

The company's model realizes the principles shared with IID, XP, and SCRUM as follows:

Iterations and Increments: Each projects develops an increment and delivers it to the LSV, the LSV being the new version of the product after integrating the increment. The projects developing the increments are run in an iterative manner.

Internal and External Releases: Software products delivered and tested in the LSV can be potentially delivered to the market. Instead of delivering to the market they can be used as an input to the next internally or externally used increment.

Time Boxing: Time boxing means that projects have a pre-defined duration with a fixed deadline. In the company model the time box is set to approximately three month. Furthermore, the LSV cycles determine when a project has to finish and drop its components to the LSV.

No Change to Started Projects: If a feature is selected and the implementation realizing the feature has been started then it is finished.

Frequent Face-to-Face Interaction: Projects are realized in small teams sitting together. Each team consists of people fulfilling different roles. Furthermore, frequent team meetings are conducted in the form of stand-up meetings as used in SCRUM.

Requirements Prioritization: A prioritized requirements list where the highest prioritized requirements are taken from the top and implemented first is one of the core principles of the company's model.

Frequent Integration: Within each LSV cycle the results from different projects are integrated and tested. As the cycles have fixed time frames frequent integration is assured.

Overall it is visible that the model shares nearly all principles with IID and realizes a majority of XP and SCRUM principles. However, we would like to point out that when comparing the results with models investigated in empirical research it is not always made explicit to what degree different practices are fulfilled in those studies. In other words, it is unknown to what extent a so-called XP-study actually implements all XP practices. This issue and its implications are further discussed in Section 5.6.

5.4 Research Method

The research method used is case study. The design of the study follows the guidelines provided for case study research in [20].

5.4.1 Case Study Context

As a complement to the process model description, the context of the study is as follows. Ericsson AB is a leading and global company offering solutions in the area of telecommunication and multimedia. Such solutions include charging systems for mobile phones, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development

model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the customer will be. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries.

5.4.2 Research Questions and Propositions

This study aims at answering the following research questions:

- *RQ1: What are the advantages and issues in industrial large-scale software development informed by agile and incremental practices?* So far, very little is known about advantages and issues of using agile and incremental practices in large-scale industrial software development. Thus, the answer to this research question makes an important step toward filling this research gap.
- *RQ2: What are the differences and similarities between state of the art and the case study results?* By answering this research question new insights in comparison to what has been studied before become explicit. Furthermore, contradictions and confirmations of previous results are made explicit and facilitate the generalizability of results.

Furthermore, propositions are stated which are similar to hypotheses, stating what the expected outcome of a study is. Propositions also help in identifying proper cases and units of analysis. The proposition is stated for the outcome of RQ2: As the case differs from those presented in state of the art new issues and benefits are discovered that have not been empirically identified before.

5.4.3 Case Selection and Units of Analysis

Three subsystems that are part of a large-scale product are studied at the company. The large-scale product is the case being studied while the three subsystems are distinct units of analysis embedded in the case. Table 5.4 summarizes some characteristics of the case and units of analysis. The LOC measure only includes code produced at the company (excluding third-party libraries). Furthermore, the approximate number of persons involved in each subsystem are stated. A comparison between the case and the Apache web server shows that the case and its units of analysis can be considered large-scale, the overall system being 20 times larger than Apache.

5.4.4 Data Collection Procedures

The data is collected through interviews and from process documentation.

Table 5.4: Units of Analysis

	Language	Size (LOC)	No. Persons
Overall System		>5,000,000	-
Subsystem 1	C++	300,000	43
Subsystem 2	C++	850,000	53
Subsystem 3	Java	24,000	17
Apache	C++	220,000	90

Selection of Interviewees

The interviewees were selected so that the overall development life cycle is covered, from requirements to testing and product packaging. Furthermore, each role in the development process should be represented by at least two persons if possible. The selection of interviewees was done as follows:

1. A complete list of people available for each subsystem was provided by management.

2. At least two persons from each role have been randomly selected from the list. The more persons are available for one role the more persons have been selected. The reason for doing so is to not disturb the projects, that is if only one person is available in a key role it disturbs the project more to occupy that person compared to when several people share the same role.

3. The selected interviewees received an e-mail explaining why they have been selected for the study. Furthermore, the mail contained information of the purpose of the study and an invitation for the interview. Overall, 44 persons have been contacted of which 33 accepted the invitation.

The distribution of people between different roles and the three subsystems (S1-S3) is shown in Table 5.5. The roles are divided into “What”, “When”, “How”, “Quality Assurance”, and “Life Cycle Management”.

What: This group is concerned with the decision of what to develop and includes people from strategic product management, technical managers and system managers. Their responsibility is to document high-level requirements and breaking them down for design and development.

When: People in this group plan the time-line of software development from a technical and project management perspective.

How: Here, the architecture is defined and the actual implementation of the system takes place. In addition, developers do testing of their own code (unit tests).

Quality Assurance: Quality assurance is responsible for testing the software and reviewing documentation.

Life Cycle Management: This includes all activities supporting the overall development process, like configuration management, maintenance and support, and packaging and shipment of the product.

Table 5.5: Distribution of Interviewees Between Roles and Units of Analysis

	S1	S2	S3	Total
What (Requirements)	2	1	1	4
When (Project Planning)	3	2	1	6
How (Implementation)	3	2	1	6
Quality Assurance	4	3	-	7
Life Cycle Management	6	4	-	10
Total	18	12	3	33

Interview Design

The interview consists of five parts, the duration of the interviews was set to approximately one hour. In the first part of the interview the interviewees were provided with an introduction to the purpose of the study and explanation why they have been selected. The second part comprised questions regarding the interviewees background, experience, and current activities. Thereafter, the actual issues and advantages were collected through a semi-structured interview. The interview was designed to collect issues and advantages from the interviewees. The interview was initially designed to only capture issues, however, during the course of the interview advantages were mentioned by the interviewees and follow-up questions were asked. In order to collect as many issues as possible, the questions have been asked from three perspectives: bottlenecks, rework, and unnecessary work. The interviewees should always state what kind of bottleneck, rework, or unnecessary work they experienced, what caused it, and where it was located in the process. The interview guide is provided in Appendix A.

Process Documentation

The company provides process documentation to their employees, as well as presentations on the process for training purposes. We study this documentation to facilitate a good understanding of the process in the organization. Furthermore, presentations given at meetings are investigated, which show the progress and first results of introducing agile and incremental practices from a management perspective. However, the main source of information is the interviews, with the process documentation mainly used to get a better understanding of the process and to triangulate what has been said in the interviews. The documentation and talking to people in the organization resulted in the description of the process model in Section 5.3.

5.4.5 Data Analysis Approach

As mentioned earlier, the conclusions of the case study are based on the mapping of the company's model to general process models, the state of the art, and the case study investigating issues and advantages.

State of the Art: In order to identify from literature which issues and advantages exist, the systematic review on agile methods [4] is used as an input. As a starting point the advantages and disadvantages have been extracted from the review (SotA). To identify more advantages and issues, the results and discussion sections of the identified papers in the review have been read, focusing on qualitative results as those are best comparable to the outcome of this study.

Process Mapping: The mapping was done based in the information gathered in the interviews, documentation of the development process, and validation with a process expert at the company. The process expert is a driver for agile implementation at the company and has profound knowledge of general agile models as well as the company's model.

Advantages/issues Mapping: The derivation of advantages and issues is done in a similar way and advantages/issues is here referred to as factors. As part of the case study analysis, the first author of the chapter transcribed more than 30 hours of audio recordings from the interviews which are used for the data analysis. The data was analyzed in a four-step process, the first four steps being conducted by the first author over a three-month period.

1. *Clustering:* The raw data from the transcriptions is clustered, grouping statements belonging together. For example, statements related to requirements engineering are grouped together. Thereafter, statements addressing similar areas are grouped. To provide an example, three statements related to requirements prioritization are shown in the text-box below.

Statement 1: The prioritization is very very hard. I do not envy the SPMs but that is the single most critical thing to get the incremental and agile process working.
Statement 2: The priority change and to inform the project that this has changed has been difficult. To solve this we have invited the release program manager who is responsible for the project to sit in and the main technical coordinator so they are part of the decision to change it. Prior to that we did not have it and we had more difficult, we just did that a couple of weeks ago, this improved the situation but still we have difficulties to have a formalized way of doing these because changes happen.
Statement 3: Theoretically, the priority list is nice. The problem is that there is a lot of changes in the situation where we are now, there are a lot of changes in the priority list here which means that we have been wasting some work done here, a little bit more than some.

2. *Derivation of Factors:* The raw data contains detailed explanations and therefore is abstracted by deriving factors from the raw data. Each factor is shortly explained in one or two sentences. The result was a high number of factors, where factors varied in their abstraction level and could be further clustered. Based on the original statements regarding the requirements prioritization the following factors (in this case issues) have been derived:

   Prioritization Issue 1: The prioritization is success critical in incremental and agile development and at the same time hard to create and maintain (based on statement 1).
   Prioritization Issue 2: Informing the project that the priorities of requirements change has been difficult and requires a more formal process (based on statement 2).
   Prioritization Issue 3: The priority list changes due to that there is a lot of changes in the situation (adoption) leading to rework (based on statement 3).

3. *Mapping of Factors:* The factors were grouped based on their relation to each other and their abstraction level in a mind map. Factors with higher abstraction level are closer to the center of the mind map than factors with lower abstraction level. In the example, the issues related to requirements prioritization are in one branch (see Figure 5.3). This branch resulted in issue CI02 in Table 5.7.

4. *Validation of Factors:* In studies of qualitative nature there is always a risk that the data is biased by the interpretation of the researcher. Thus, the factors have

been validated in two workshops with three representatives from the company. The representatives have an in-depth knowledge of the processes. Together, the first three steps of analysis described here were reproduced with the authors and company representatives. As input for the reproduction of factors, a subset of randomly selected issues and advantages have been selected. The outcome of the workshop was positive as there was no disagreement on the interpretation of the factors. To further improve the data the workshop participants reviewed the final list of issues and advantages and only provided small improvement suggestions on how to formulate them. Thus, the list of factors can be considered being of high quality.

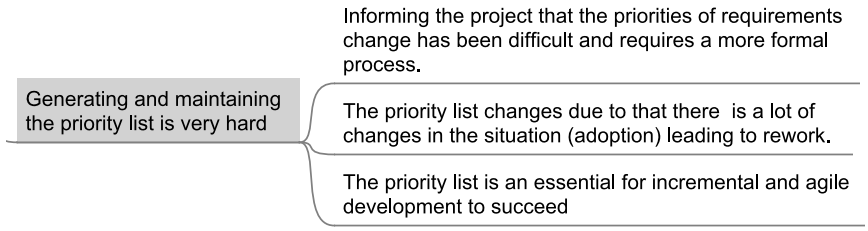


Figure 5.3: Cutout from Mind Map

Finally, the SotA and case study results are compared to identify whether new issues have been identified in this case study, and to explain why other advantages found in SotA cannot be seen in the case study. It is important to mention that not all issues and advantages found in the case study are considered in the comparison. Only general issues and advantages should be taken into consideration. Thus, we only included issues that have been mentioned by two or more persons.

5.4.6 Threats to Validity

Threats to the validity of the outcome of the study are important to consider during the design of the study, allowing actions to be taken mitigating them. Threats to validity in case study research are reported in [20]. The threats to validity can be divided into four types: construct validity, internal validity, external validity and reliability.

Construct Validity: Construct validity is concerned with obtaining the right measures for the concept being studies. One threat is the selection of people to obtain the appropriate sample for answering the research questions. Therefore, experienced people from the company selected a pool of interviewees as they know the persons

and organization best. From this pool the random sample was taken. The selection by the representatives of the company was done having the following aspects in mind: process knowledge, roles, distribution across subsystem components, and having a sufficient number of people involved (although balancing against costs). Furthermore, it is a threat that the presence of the researcher influences the outcome of the study. The threat is reduced as there has been a long cooperation between the company and university and the author collecting the data is also employed by the company and not viewed as being external. Construct validity is also threatened if interview questions are misunderstood or misinterpreted. To mitigate the threat pre-tests of the interview have been conducted.

Internal Validity: Internal validity is primarily for explanatory and causal studies, where the objective is to establish a causal relationship. As this study is of exploratory nature internal validity is not considered.

External Validity: External validity is the ability to generalize the findings to a specific context. It is impossible to collect data for a general process, i.e. exactly as it is described in literature. The process studied is an adoption of practices from different general process models (see Section 5.3). Care has been taken to draw conclusions and map results to these general models to draw general conclusions and not solely discussing issues that are present due to the specific instantiation of the process at the studied setting. However, if one maps the general findings in this chapter to other development processes their context must be taken into account. Furthermore, a potential threat is that the actual case study is conducted within one company. To minimize the influence of the study being conducted at one company, the objective is to map the findings from the company specific processes and issues to general processes. The characteristics of the context and practices used in the process are made explicit to ease the mapping (see Table 5.3).

Reliability: This threat is concerned with repetition or replication, and in particular that the same result would be found if re-doing the study in the same setting. There is always a risk that the outcome of the study is affected by the interpretation of the researcher. To mitigate this threat, the study has been designed so that data is collected from different sources, i.e. to conduct triangulation to ensure the correctness of the findings. The interviews have been recorded and the correct interpretation of the data has been validated through workshops with representatives of the company.

5.5 Results

First, the advantages identified in the case study are compared with SotA, and the same is done for the issues.

5.5.1 Advantages

Table 5.6 shows the advantages identified in the case study, furthermore the ID of the advantages of SotA clearly related to the ones in the case study are stated in column SotA (ID). It is shown that six out of eight advantages can be clearly linked to those identified in literature.

Transparency and Control: Better control and transparency is achieved by having small and manageable tasks (A04). The case study led to the same result. The prioritized list of requirements consists of requirements packages that have to be implemented and the requirements packages have a small scope (for example compared to waterfall models where the complete scope is defined upfront). Due to clear separation of packages which are delivered as an increment, responsibilities for an increment can be clearly defined increasing transparency (CA03). In particular problems and successes are more transparent. That is, if an increment is dropped to the LSV for test one can trace which increments are of high or low quality and who is responsible for them. Consequently, this creates incentives for teams to deliver high quality as their work result is visibly linked to them (CA08).

Learning, Understanding, and other Benefits of Face-to-Face Communication: In agile development team members communicate intensively face-to-face as they have frequent meetings and are physically located together (A07). Thus, learning and understanding from each other is intensified. In the case study, the interviewees provided a concrete example for this. Before using agile, testers and designers were separated. Consequently designers were not able to put themselves in the shoes of the testers verifying their software or understand what information or documentation would help testing. Now designers and testers sit together and thus they can learn from each other. The designers understand how the quality of the implementation impacts the testers. Furthermore, testers can point designers to parts of the system that from their perspective are critical and thus require more intensive testing (CA04). The direct communication also enables instant testing due to short lines of communication (CA07). An additional benefit is the increased informal communication where important information is continuously shared, ultimately resulting in less rework and higher quality (CA06).

Frequent Feedback for each Iteration: Knowledge is transfered through frequent feedback whenever completing and delivering an iteration (A01). In the case study a similar result was obtained. Whenever increments are dropped to the LSV there is a clear visibility of who delivered what and with what level of quality. The frequency of integration is enforced by pre-defined LSV cycles that require integration every few weeks. This of course also facilitates frequent feedback (CA04).

Further advantages that have not been explicitly identified in literature surfaced during the case study.

Low Requirements Volatility: Small requirements packages are prioritized and can go quickly into the development due to their limited scope. When implemented they are dropped to the LSV and can potentially be released to the market. As the market in this case is highly dynamic this is an important advantage. That is, if hot requirements can be implemented quickly and thus can be released before the customers' needs change (CA01).

Work Started is always Completed: Packages that have started implementation are always completed. Therefore, there is very little waste in development as work done is not discarded, but ends up as a running part of the software. However, it should be emphasized that it is essential to implement the right things, making requirements prioritization an essential issue for this advantage to pay off (CA02).

Table 5.6: Advantages Identified in Case Study

ID	Advantages	SotA (ID)
CA01	Small projects allow to implement and release requirements packages fast which leads to reduction of requirements volatility in projects.	
CA02	The waste of not used work (requirements documented, components implemented etc.) is reduced as small packages started are always implemented.	
CA03	Requirements in requirements packages are precise and due to the small scope estimates for the package are accurate.	A04
CA04	Small teams with people having different roles only require small amounts of documentation as it is replaced with direct communication facilitating learning and understanding for each other.	A07
CA05	Frequent integration and deliveries to subsystem test (LSV) allows design to receive early and frequent feedback on their work.	A01
CA06	Rework caused by faults is reduced as testing priorities are made more clear due to prioritized features, and that testers as well as designers work closely together.	A07
CA07	Time of testers is used more efficiently as in small teams as testing and design can be easily parallelized due to short ways of communication between designers and testers (instant testing).	A07
CA08	Testing in the LSV makes problems and successes transparent (testing and integration per package) and thus generates high incentives for designers to deliver high quality.	A04

5.5.2 Issues

The issues identified in this case study as well as the references to similar issues of SotA are shown in Table 5.7. The following issues are shared between SotA and the findings of this study.

Testing Lead Times and Maintenance: The realization of continuous testing with a variety of platforms and test environments is challenging and requires much effort (I01). This SotA issue relates to two issues identified in this case study. First, testing lead times are extended as packages that should be delivered to the LSV might not be dropped due quality issues or that the project is late. If this happens shortly before an LSV cycle ends and the next increment is built, the package has to wait for the whole next cycle to be integrated (CI07). Secondly, if increments are released more frequently maintenance effort increases. That is, customers report faults for many different versions of the software making it harder to reproduce the fault on the right software version as well as in the right testing environment including released hardware (CI07).

Management Overhead and Coordination: Agile methods do not scale well (I03). In fact, we found that it is challenging to make agile methods scalable. On the one hand, small projects can be better controlled and results are better traceable (as discussed for CA08). On the other hand, many small projects working toward the same goal require much coordination and management effort. This includes planning of the technical structure and matching it against a time-line for project planning (CI11).

Little Focus on Architecture: The architecture receives little focus in agile development leading to bad design decisions (I02). The company's development model requires a high level architecture plan (anatomy plan) enabling them to plan the time-line of the projects. However, dependencies between parts of the system rooted in technical details are not covered in the plan. As one project implementing a specific package has no control over other packages the discovery of those dependencies early has not been possible (CI12).

Further issues that have not been explicitly identified in literature surfaced during the case study.

Requirements Prioritization and Handover: In the development of large scale products the strategy of the product and the release plans have to be carefully planned and involve a high number of people. Due to the complexity and the number of people that have to be involved in each decision the continuity of the requirements flow is thwarted (CI01). Consequently teams have to wait for requirements and a backlog is created in development (CI03). The decision is further complicated by prioritization, prioritization being perceived as an essential success factor by the interviewees, which also plays an important role in other agile methods. For example, SCRUM uses

a product backlog which is an ordered list of features, the feature of highest priority always being at the top of the list. Getting the priority list right is challenging as the requirements list in itself has to be agile reflecting changing customer needs (dynamic re-prioritization)(C2).

Test Coverage Reduction of Basic Test: Teams have to conduct unit testing and test their overall package before delivering to the LSV. However, the concept of small projects and the lack of independent verification make it necessary that the LSV compensates the missing test coverage. The perception of interviewees was that it is hard to squeeze the scope into three month projects. One further factor is the get-together of designers and testers resulting in dependent verification and validation. For example, designers can influence testers to only focus on parts of the system, saying that other parts do not have to be tested because they did not touch them.

Increased Configuration Management Effort: Configuration management has to coordinate a high number of internal releases. Each LSV is a baseline that could be potentially released to the market. Thus, the number of baselines in agile development is very high.

Issues CI05, CI09, and CI10 are more related to the context than the other issues described earlier, even though from the company's perspective they play an important role and thus have been mentioned by several people. Because of the limited generalizability of those issues to other models they are only discussed briefly.

Due to the previous way of working at the company a high amount of documentation remained (CI05). The ambition is to reduce the number of documents as many documents are unnecessary because they are quickly outdated while other documents can be replaced by direct communication (CA04). However, this issue could be generalized to other companies in transition to a more agile way of working. Issues (CI09) and (CI08) are related to product packaging which mainly focuses on programming the configuration environment of the system. The environment allows to select features for specific customers to tailor the products to their specific needs (product customization). The findings are that this requires long lead times (CI09) and that product packaging gets information too late, even though they could start earlier (CI10).

5.6 Discussion

This section discusses the comparison of state of the art and the case study results. We describe the observations made based on the results, and the implications for practice and research. This includes suggestions for future work.

Table 5.7: Issues Identified in Case Study

ID	Issue	SotA (ID)
CI01	Handover from requirements to design takes time due to complex decision processes.	
CI02	The priority list is essential in the company's model to work and is hard to create and maintain.	
CI03	Design has free capacity due to the long lead times as in requirements engineering complex decision making (e.g, due to CI02) takes long time.	
CI04	Test coverage reduction within projects due to lack of independent testing and shortage of projects, requiring LSV to compensate coverage.	
CI05	The company's process requires to produce too much testing documentation.	
CI06	LSV cycle times may extend lead-time for package deliveries as if a package is not ready or rejected by testing it has to wait for the next cycle.	I01
CI07	Making use of the ability of releasing many releases to the market increases maintenance effort as many different versions have to be supported and test environments for different versions have to be recreated.	I01
CI08	Configuration management requires high effort to coordinate the high number of internal releases.	
CI09	The development of the configuration environment to select features for customizing solutions takes a long time due to late start of product packaging work and use of sequential programming libraries.	
CI10	Product packaging effort is increased as it is still viewed from a technical point of view, but not from a commercial point of view.	
CI11	Management overhead due to a high number of teams requiring much coordination and communication between.	I03
CI12	Dependencies rooted in implementation details are hard to identify and not covered in the anatomy plan.	I02

5.6.1 Practices Lead to Advantages and Issues

Observation: Using certain practices bring benefit and at the same time raise different issues. In related work this was visible for outcomes related to pair programming.

On one hand it facilitates learning, but on the other hand it is also exhaustive and leads to problems if the programmers are on different levels. Similar results have been identified in this case study as well:

- Small projects increase control over the project, increase transparency, and effort can be estimated in a better way (CA08). At the same time the small projects have to be coordinated which raises new challenges from a management perspective with large scale in terms of size of product and people involved (CI11).
- Frequent integration and deliveries to the LSV in given cycles provide regular feedback to the developers creating packages (A01). Though related issues are that if a package is rejected it has to wait for the whole new LSV cycle (CI06) and configuration management has increased work effort related to baselining (CI08).
- Direct communication facilitates learning and understanding for each other (CA04). However, the close relation between testers and designers affects independent testing negatively (CI04).

Implications for Practice: For practice this result implies that companies have to choose practices carefully, not only focusing on the advantages that come with the techniques. At the same time it is important to be aware of drawbacks using incremental and agile practices which seem to be overlooked all too often.

Implications for Research and Future Work: Research has to support practice in raising the awareness of problems related to incremental and agile development. None of the studies in the systematic review by Dybå et al. [4] had the identification of issues and problems as the main study focus. To address this research gap we propose to conduct further qualitative studies focusing on issues which often seem to come together with the advantages. It is also important to find solutions solving the issues in order to exploit the benefits that come with agile to an even greater degree. This requires new methods to fully utilize the benefits of agile, to name a few general areas that should be focused on:

- Agile requirements prioritization techniques to support and deal with frequent changes in priority lists which have been identified as success critical (see CI02).
- Research on tailoring of configuration management for agile due to high number of baselines and changes that need to be maintained (see CI08).
- Research on decision making processes and decision support in agile processes (see CI01).

5.6.2 Similarities and Differences between SotA and Industrial Case Study

Observation: The initial proposition was that there is a difference in issues between what is said in SotA and the findings of the case study. The opposite is true for the advantages, we found that there is quite a high overlap between advantages identified in SotA and this case study. Six out of eight advantages have also been identified in SotA as discussed in Section 5.5. This is an indication that agile leads to similar benefits in large scale development and small scale development. On the other hand, the overlap regarding the issues is smaller. Many issues identified in SotA are not found in this case study, mainly because a few of them are related to pair programming which is not a principle that is applied yet at the company. On the other hand, only a few issues (three out of twelve) identified in this case study have been empirically shown in other studies. Several explanations are possible for this result. Firstly, the studies did not have issue identification as a main focus. Another explanation is that even though agile leads to benefits in large-scale development it is also harder to implement due to increased complexity in terms of product size, people and number of projects (reflected in issues like CI01, CI02, CI03, CI08, CI11), which of course results in more issues raised.

Implications for Practice: Many of the new problems found in the case study occur due to complexity in decision making, coordination, and communication. We believe that when studying a company developing small products then the same benefits would be found, but the number of issues identified would be much lower. Thus, companies in large-scale development which intend to adopt incremental and agile methods need to be aware of methods supporting in handling the complexity. For example, Cataldo et al. [2] propose a method that helps coordinate work based on the automatic identification of technical dependencies, i.e. this makes more clear which teams have to communicate with each other.

Implications for Research: This observation leads to the same conclusion as the previous one (practices lead to advantages and issues): further knowledge is needed about what are the main issues in large scale agile development and how they can be addressed to get the most out of the benefits.

5.6.3 A Research Framework for Empirical Studies on Agile Development

The need for a research framework is an important implication for research. That is, in order to learn more about issues and make different studies comparable we believe that there is a great need for a framework of empirical studies on agile development. For

example, when agile is studied it is often not clear how a certain model is implemented and to what degree the practices are fulfilled. Instead, it is simply said that XP or SCRUM is studied. However, from our experience in industry we know that methods presented in books are often tailored to specific needs and that practices are enriched or left out as they do not fit into the context of the company. This makes it hard to determine which practices or combinations of practices in a given context lead to advantages or issues. Such a framework could include information about:

- Attributes that should be provided in order to describe the context. For example, studies do not report the domain they are investigating or how many people are involved in the development of the system (see for example Table 5.2). Furthermore, product complexity should be described and it needs to be clear whether a team or product has been studied.
- Practices should be made explicit and it should be explained how and to what degree they are implemented allowing the reader to generalize the outcome of the studies. For example, the framework should describe when a practice is considered as fully, partly, or not at all fulfilled.

5.7 Conclusions and Future Work

This chapter compares the state of the art investigating issues and advantages when using agile and incremental development models with an industrial case study where agile as well as incremental practices are applied. The articles considered in the state of the art are based on empirical studies. The case being studied can be characterized as large-scale in terms of product size and number of persons involved in the development process. Regarding the research questions and contributions we can conclude:

Issues and Advantages: We found that implementing agile and incremental practices in large-scale software development leads to benefits in one part of the process, while raising issues in another part of the process. For example, using small and coherent teams increases control over the project, but leads to new issues on the management level where the coordination of the projects has to take place. Further examples for this have been identified in the study.

Comparison of State of the Art and Case Study - Advantages: Previous empirical studies and the case study results have a high overlap for the advantages. In summary, the main advantages agreed on by literature this case study are 1) requirements are more precise due to reduced scope and thus easier to estimate, 2) direct communication in teams reduces need for documentation, 3) early feedback due to frequent deliveries, 4) rework reduction, 5) testing resources are used more efficiently, and 6)

higher transparency of who is responsible for what creates incentives to deliver higher quality. New advantages identified in this case study are 1) low requirements volatility in projects, and 2) reduction of waste (discarded requirements) in the requirements engineering process.

Comparison of State of the Art and Case Study - Issues: Only few issues identified in the case study are mentioned in literature. Issues agreed on are 1) challenges in regard to realize continuous testing, 2) increased maintenance effort with increase of the number of releases, 3) management overhead due to the need of coordination between teams, and 4) detailed dependencies are not discovered on detailed level due to lack of focus on design. In total eight new issues have been identified in this case study, five are of general nature while three are strongly related to the study context. The general issues are 1) Long requirements engineering duration due to complex decision processes in requirements engineering, 2) requirements priority lists are hard to create and maintain, 3) Waiting times in the process, specifically in design waiting for requirements, 4) reduction of test coverage due to shortage of projects and lack of independent testing, 5) increased configuration management effort. The three context related issues are 6) high amount of testing documentation, 7) long durations for developing the configuration environment realizing product customizations, and 8) increase in product-packaging effort.

The proposition of the study is partly true, i.e. the study did not identify many new advantages that have not been found in previous empirical studies. However, the study identified new issues that have not been reported in empirical studies before. Those issues are mainly related to increased complexity when scaling agile.

Furthermore, we identified the need for an empirical research framework for agile methods which should help to make studies comparable. In future work more qualitative studies with an explicit focus on issue identification have to be conducted.

5.8 References

[1] Bouchaib Bahli and El-Sayed Abou-Zeid. The role of knowledge creation in adopting XP programming model: an empirical study. In *ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society*, 2005.

[2] Marcelo Cataldo and Patrick Wagstrom and James D. Herbsleb and Kathleen M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proc. of the Conference on Computer Supported Cooperative Work (CSCW 2006)*, pages 353–362, 2006.

[3] David Cohen, Mikael Lindvall, and atricia Costa. *Advances in Computers, Advances in Software Engineering*, chapter An Introduction to Agile Methods. Elsevier, Amsterdam, 2004.

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

[5] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an agile methodology implementation. In *Proc. of the 30th EUROMICRO Conference (EUROMICRO 2004)*, pages 326–333, 2004.

[6] Daniel Karlström and Per Runeson. Combining agile methods with stage-gate project management. *IEEE Software*, 22(3):43–49, 2005.

[7] Craig Larman. *Agile and iterative development: a manager's guide*. Pearson Education, 2003.

[8] Adrian MacKenzie and Simon R. Monk. From cards to code: how extreme programming re-embodies programming as a collective practice. *Computer Supported Cooperative Work*, 13(1):91–117, 2004.

[9] Chris Mann and Frank Maurer. A case study on the impact of Scrum on overtime and customer satisfaction. In *Proc. of the AGILE Conference (AGILE 2005)*, pages 70–79, 2005.

[10] Katiuscia Mannaro, Marco Melis, and Michele Marchesi. Empirical analysis on the satisfaction of it employees comparing XP practices with other software development methodologies. In *Proc. of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, pages 166–174, 2004.

[11] Angela Martin, Robert Biddle, and James Noble. The XP customer role in practice: Three studies. In *Agile Development Conference*, pages 42–54, 2004.

[12] Pete McBreen. *Questioning extreme programming*. Pearson Education, Boston, MA, USA, 2003.

[13] Grigori Melnik and Frank Maurer. Perceptions of agile practices: a student survey. In *Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods (XP/Agile Universe 2002*, pages 241–250, 2002.

[14] H. Merisalo-Rantanen, Tuure Tuunanen, and Matti Rossi. Is extreme programming just old wine in new bottles: a comparison of two cases. *J. Database Manag.*, 16(4):41–61, 2005.

[15] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Softw. Engg.*, 13(3):303–337, 2008.

[16] Hugh Robinson and Helen Sharp. The characteristics of XP teams. In *Proc. of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004)*, pages 139–147, 2004.

[17] Matt Stephens and Doug Rosenberg. *Extreme programming refactored: the case against XP*. Apress, Berkeley, CA, 2003.

[18] Harald Svensson and Martin Höst. Introducing an agile process in a software maintenance and evolution organization. In *Proc. of the 9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, pages 256–264, 2005.

[19] Bjørnar Tessem. Experiences in learning XP practices: A qualitative study. In *Proc. of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004)*, pages 131–137, 2003.

[20] Robert K. Yin. *Case study research: design and methods, 3rd Edition, Applied Social Research Methods Series, Vol. 5*. Prentice Hall, 2002.

Chapter 6

An Empirical Study of Lead-Times in Incremental and Agile Software Development

Kai Petersen
To Appear in Proceedings of the International Conference on Software Process (ICSP 2010)

6.1 Introduction

Lead-time (also referred to as cycle-times) is the time it takes to process an order from the request till the delivery [6]. An analysis and improvement of lead-time is highly relevant. Not being able to deliver in short lead-times leads to a number of disadvantages on the market, identified in the study of Bratthall et al. [4]: (1) The risk of market lock-out is reduced [12]. Bratthall et al. [4] provided a concrete example for that where one of the interviewees reported that they had to stall the introduction of a new product because the competitor was introducing a similar product one week earlier; (2) An early enrollment of a new product increase probability of market dominance [14]. One of the participants in the study of Bratthall et al. [4] reported that due to introducing a product three months after a competitor the company is holding 30 % less of the world

market in comparison to the market leader; (3) Another benefit of being early on the market is that the product conforms more to the expectations of the market [13]. This is due to the market dynamics. Petersen et al. [10] found that a large portion (26 %) of gathered requirements are already discarded during development. Furthermore, the long lead-time provides a time-window for change requests and rework.

The review of literature revealed that, to the best of our knowledge, an empirical analysis of lead-times in incremental and agile development has not been conducted so far. However, as there is an increasing number of companies employing incremental and agile practices it is important to understand lead-time behavior. The studied company intended to determine target levels for lead-times. The open question at the company was whether requirements should have different target levels depending on the following factors:

1. The distribution of lead-times between different phases.

2. The impact a requirement has on the systems. The impact is measured in terms of number of affected systems. Here we distinguish between single-system requirements (a requirement only affects one system) and multi-system requirements (a requirement affects at least two systems).

3. The size of the requirements.

This study investigated the effect of the three factors on lead-time. It is important to stress that existing work indicates what outcomes can be expected for the different factors, the expected results being presented in the related work. However, the outcome to be expected was not clear to the practitioners in the studied company. Hence, this study sets out with formulating a set of hypotheses related to the factors without assuming a specific outcome of the hypotheses prior to analyzing them.

The research method used was an industrial case study of a company developing large-scale systems in the telecommunication domain. The quantitative data was collected from a company proprietary system keeping track of the requirements flow throughout the software development lifecycle.

The remainder of the paper is structured as follows. Section 6.2 presents related work. The research method is explained in Section 6.3. The results of the study are shown in Section 6.4. Section 6.5 discusses the results. Section 6.6 concludes the paper.

6.2 Related Work

Chapter 3 presents lead-times for waterfall development, showing that the majority of the time (41 %) is spent on requirements engineering activities. The remaining time was distributed as follows: 17 % in design and implementation, 19 % on verification, and 23 % on the release project. As in agile software development the main activitis should be coding and testing [3] the literature would suggest that those are the most time consuming activities.

Petersen and Wohlin [8] investigated issues hindering the performance of incremental and agile development. When scaling agile the main issues are (1) complex decision making in the requirements phase; (2) dependencies of complex systems are not discovered early on; and (3) agile does not scale well as complex architecture requires up-front planning (see also Chapter 5). Given this qualitative result the literature indicates that with increase of requirements impact the lead-time should increase. For example, if a requirement can only be deliverable when parts of it are implemented across several systems a delay in one system would lead to prolonged lead-times for this requirement.

Harter et al. [7] identified that lines of code (size) is a predictor for cycle time. This was confirmed by [1] who found that size was the only predictor for lead-time. Hence, from the related work point of view an increase of size should lead to an increase of lead-time.

Collier [2] summarizes a number of issues in cycle time reduction and states: (1) size prolongs lead-time, and (2) dependencies influence lead-time.

Carmel [5] investigated key success factors for achieving short lead-times. The finding shows that team factors (small team size, cross-functional teams, motivation) are critical. Furthermore, an awareness of lead-times is important to choose actions specifically targeted towards lead-time reduction. However, it is important to take quality into consideration when taking actions towards lead-time reduction.

None of the lead-time studies focuses on agile development, and hence raising the need for empirical studies on lead-time in an incremental and agile development context.

6.3 Research Method

The research method used was a quantitative case study, the case being the telecommunication company Ericsson AB. The systems studied were developed in Sweden and India.

6.3.1 Research Context

The research context is important to describe in order to know to what degree the results of the study can be generalized [9]. Table 6.1 shows the context elements for this study. The analysis focused on in total 12 systems of which 3 systems are independent. The remaining nine systems belong to a very large communication system and are highly dependent on each other. Thus, all requirements belonging to the independent systems are treated as single-system requirements. The same applies to requirements only affecting one of the nine dependent systems.

Table 6.1: Context Elements

Element	Description
Maturity	All systems older than 5 years
Size	Large-scale system with more than 5,000,000 LOC
Number of systems	9 dependent systems (indexed as A to I) and 3 independent (indexed as J to K)
Domain	Telecommunication
Market	Highly dynamic and customized market
Process	On the principle level incremental process with agile practices in development teams
Certification	ISO 9001:2000
Practices	Continuous integration; Internal and external releases; Time-boxing with sprints; Face-to-face interaction (stand-up meetings, co-located teams); Requirements prioritization with metaphors and Detailed requirements (Digital Product Backlog); Refactoring and system improvements

The process of the company is shown in Figure 6.1. Requirements from the market are prioritized and described as high level requirements (HLR) in form of metaphors. These are further detailed by cross-functional work teams (people knowing the market and people knowing the technology) to detailed requirements specifications (DRS). The anatomy of the system is used to identify the impact of the HLR on the system. The impact determines how many systems are affected by the requirement. A requirement affecting one system is referred to as single-system requirement, while a requirement affecting multiple system is referred to as a multi-system requirement. Within system development agile teams (ATs) implement and unit test the requirements within four week sprints. The teams deliver the requirements to the system level test to continu-

ously verify the integration on system level every four weeks. Furthermore, the system development delivers their increments to the compound system test, which is also integrating in four week cycles. Requirements having passed the test are handed over to the release projects to be packaged for the market.

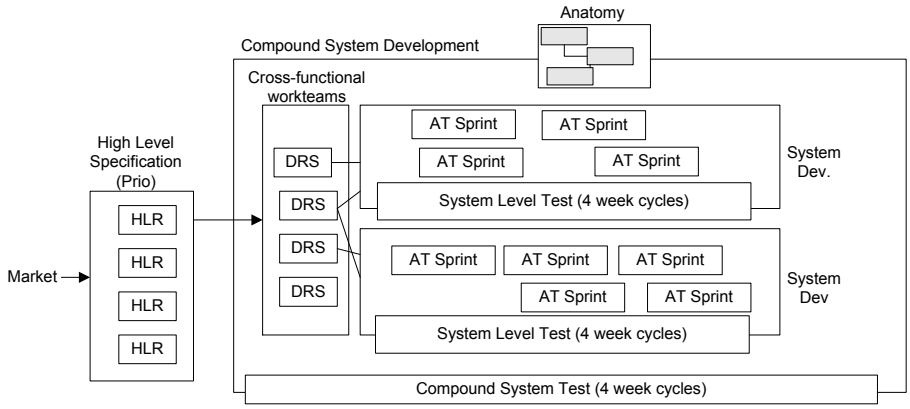


Figure 6.1: Development Process

6.3.2 Hypotheses

The hypotheses are related to differences between multi- and single-system requirements, the distribution of the lead-time between phases, and the difference between sizes of requirements. As mentioned earlier the goal is not to reject the null hypotheses, but to determine whether the different factors lead to differences in lead-time. In the case of not rejecting the null-hypotheses the factors do not affect the lead-time, while the rejection of the null-hypotheses implies that the factors effect lead-time. The following hypotheses were made:

- *Phases:* There is no difference in lead-time between phases ($H_{0,phaswe}$) opposed to there is a is a difference ($H_{a,phase}$).
- *Multi vs. Single:* There is no difference between multi- and single-system requirements ($H_{0,mult}$) opposed to there is a difference ($H_{a,mult}$).
- *Size:* There is no difference between sizes ($H_{0,size}$) opposed to there is a difference ($H_{a,size}$).

6.3.3 Data Collection

The lead-time is determined by keeping track of the duration the high level requirements reside in different states. When a certain activity related to the high-level requirement is executed (e.g. specification of the requirement) then the requirement is put into that state. For the tracking of lead-times a time-stamp was captured whenever a requirement enters a state, and leaves a state.

The lead-time data was collected from an electronic Kanban solution where the requirements can be moved between phases to change their state. The system can be edited over the web, showing the lists of the requirements and in which phase they are. Whenever a person is moving a requirement from one phase to another, a date is entered for this movement. The requirements go through the following states:

- *State Detailed Requirements Specification:* The state starts with the decision to hand over requirement to the cross-functional work-team for specification, and ends with the hand-over to the development organization.
- *State Implementation and Unit Test:* The state starts with the hand-over of the requirement to the development organization and ends with the delivery to the system test.
- *State Node/System Test:* The state starts with the hand-over of the requirement to the system/node test and ends with the successful completion of the compound system test. The time includes maintenance for fixing discovered defects.
- *State Ready for Release:* The state starts when the requirement has successfully completed the compound system test and thus is ready for release to the customer.

From the duration the requirements stay in the states the following lead-times were calculated:

- LT_a: Lead-time of a specific activity a based on the duration a requirement resided in the state related to the activity.
- LT_{n-a}: Lead-time starting with an activity a and ending with an activity n. In order to calculate this lead-time, the sum of the durations of all activities to work on a specific high-level requirement were calculated.

Waiting times are included in the lead-times. The accuracy of the measures is high as the data was under regular review due to that the electronic Kanban solution was used in daily work, and the data was subject to a monthly analysis and review.

6.3.4 Data Analysis

The descriptive statistics used were box-plots illustrating the spread of the data. The hypotheses were analyzed by identifying whether there is a relationship between the variable lead-time and the variables phases ($H_{0,phase}$) and system impact ($H_{0,mult}$). For the relationship between lead-time and system impact the Pearson correlation was used to determine whether there is a linear relationship, and the Spearman correlation to test whether there is a non-linear relationship. For the relationship between lead-time and phase ($H_{0,phase}$) no correlation was used as phase is a categorical variable. In order to capture whether phase leads to variance in lead-time we test whether specific phases lead to variance in lead-time, this is done by using stepwise regression analysis. Thereby, for each category a dummy variable is introduced.

If there is a relationship between the variables (e.g. between system impact and lead-time) this would mean that the system impact would be a variable explaining some of the variance in the lead-time. The hypotheses for size ($H_{0,size}$) was only evaluated using descriptive statistics due to the limited number of data points.

6.3.5 Threats to Validity

Four types of threats to validity are distinguished, namely conclusion validity (ability to draw conclusions about relationships based on statistical inference), internal validity (ability to isolate and identify factors affecting the studied variables without the researchers knowledge), construct validity (ability to measure the object being studied), and external validity (ability to generalize the results) [15].

Conclusion Validity: The statistical inferences that could be made from this study to a population are limited as this study investigates one particular case. In consequence, no inference statistics for comparing sample means and medians with regard to statistical significance are used. Instead correlation analysis was used, correlation analysis being much more common for observational studies such as case studies. For the test of hypotheses $H_{0,phase}$ we used stepwise regression analysis, regression analysis being a tool of statistical inference. Hence, the interpretation of regression analysis in observational studies has to be done with great care as for a single case study no random sampling with regard to a population has been conducted. The main purpose of the regression was to investigate whether the category leads to variance in lead-time at the specific company. That is, companies with similar contexts might make similar observations, but an inference to the population of companies using incremental and agile methods based on the regression would be misleading.

Internal Validity: One threat to internal validity is the objectivity of measurements, affected by the interpretation of the researcher. To reduce the risk the researcher pre-

sented the lead-time data during one meeting and discussed it with peers at the company.

Construct Validity: Reactive bias is a threat where the presence of the researcher influences the outcome of the study. The risk is low due that the researcher is employed at the company, and has been working with the company for a couple of years. Correct data is another threat to validity (in this case whether the lead-time data is accurately entered and up-to-date). As the system and the data entered support the practitioners in their daily work the data is continuously updated. Furthermore, the system provides an overview of missing values, aiding in keeping the data complete which reduces the risk.

External Validity: One risk to external validity is the focus on one company and thus the limitation in generalizing the result. To reduce the risk multiple systems were studied. Furthermore, the context of the case was described as this makes explicit to what degree the results are generalizable. In this case the results apply to large-scale software development with parallel system development and incremental deliveries to system testing. Agile practices were applied on team-level.

6.4 Results

6.4.1 Time Distribution Phases

Figure 6.2 shows the box-plots for lead-times between phases P1 (requirements specification), P2 (implementation and unit test), P3 (node and system test), and P4 (release projects). The box-plots do not provide a clear indication of the differences of lead-time distribution between phases as the box-plots show high overlaps between the phases.

Table 6.2 provides an overview of the statistical results of the correlations between phase and lead-time across systems. The stepwise regression shows that Dummy 4 was highly significant in the regression. Though, the overall explanatory power (which was slightly increased by the introduction of the Dummy 1) is still very low and accounts for 1.14 % of the variance in lead-time (R^2). Hence, $H_{0,phase}$ cannot be rejected with respect to this particular case.

6.4.2 Multi-System vs. Single-System Requirements

Figure 6.3 shows single system requirements (labeled as 0) in comparison to multi-system requirements (labeled as 1) for all phases and for the total lead-time. The box-plots indicate that there is no difference between single and multi-system requirements. In fact, it is clearly visible that the boxes and the median values are on the same level.

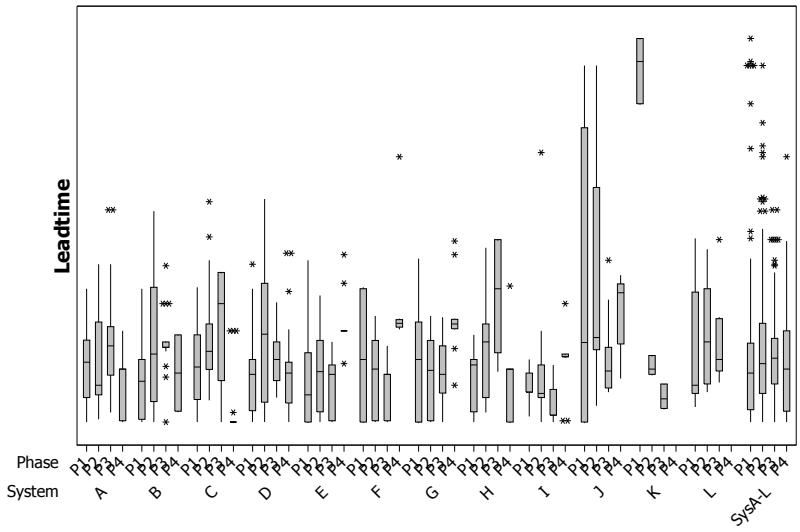


Figure 6.2: Comparison of Lead-Times between Phases

Table 6.2: Results for Distribution of Lead-Time Phases, N=823

Step	One	Two
Constant	49.91	52.03
Dummy 4 (P4)	-8.9	-11.0
t-value	-2.5	-2.94
p-value	0.013	0.003
Dummy 1 (P1)	-	-6.4
t-value	-	-1.79
p-value	-	0.074
Expl. power R^2	0.0075	0.0114

As there is no observable difference between system impacts we investigated whether the number of systems a requirement is dependent on has an influence on lead-time. As shown in the box-plots in Figure 6.4 the lead-time does not increase with a higher number of affected systems. On the single system side even more outliers to the top of the box-plot can be found for all phases.

The correlation analysis between system impact and lead-time across systems is shown in Table 6.3. The table shows that the correlations are neither close to $+/-1$,

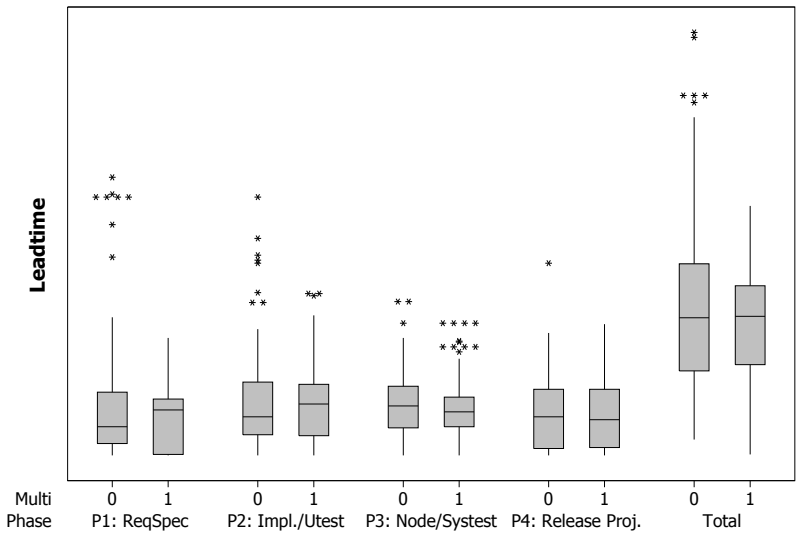


Figure 6.3: Single-System (label=0) vs. Multi-System Requirements (label=1)

and are not significant. Hence, this indicates that the two variables do not seem to be related in the case of this company, leading to a rejection of $H_{0,multi}$.

Table 6.3: Test Results for $H_{0,multi}$, N=823

Statistic	Value	p
Pearson (φ)	-0.029	0.41
Spearman (ρ)	0.074	0.57
Expl. power (R^2)	0.001	

6.4.3 Difference Between Small / Medium / Large

Figure 6.5 shows the difference of lead-time between phases grouped by size, the size being an expert estimate by requirements and system analysts. The sizes are defined as intervals in person days, where $Small(S) := [0;300]$, $Medium(M) := [301;699]$, and $Large(L) := [700;\infty]$. No correlation analysis was used for analyzing this data as three groups only have two values, namely P2-Large, P3-Medium, and P3-Large. The reason for the limitation was that only recently the requirements were attributed with the size.

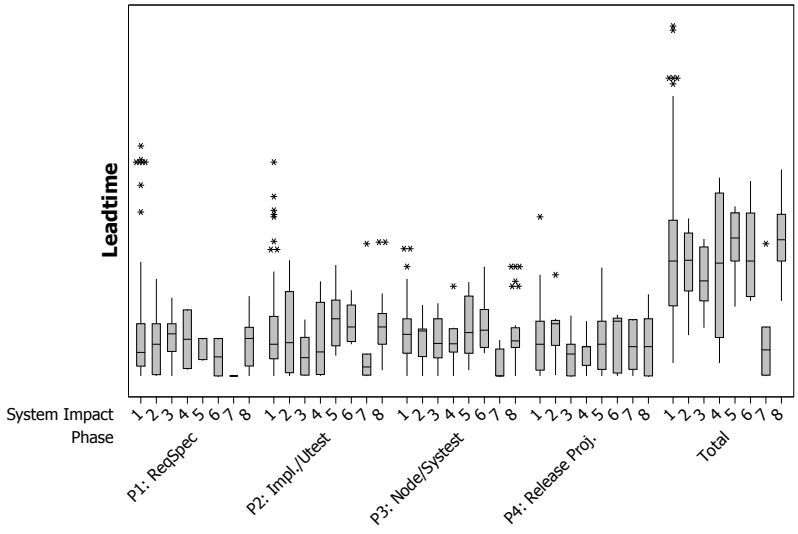


Figure 6.4: Difference for System Impact (Number of systems a requirement is affecting, ranging from 1-8)

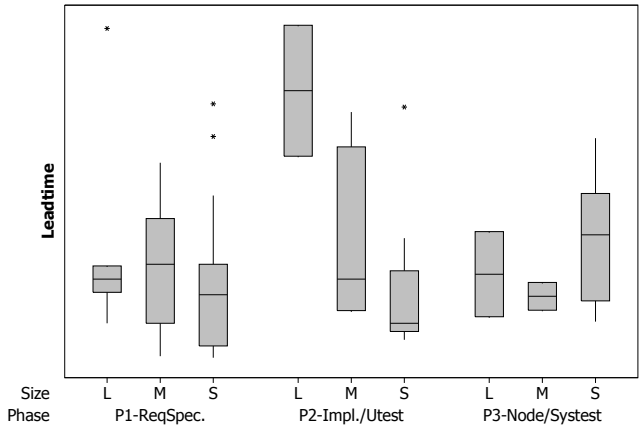


Figure 6.5: Difference for Small, Medium, and Large Requirements

However, the data already shows that the difference for size seems to be small in the phases requirements specification and node as well as compound system testing. However, the size of requirements in the design phase shows a trend of increased lead-time with increased size.

6.5 Discussion

This section presents the practical and research implications. Furthermore, the reasons for the results seen in the hypotheses tests are provided. The explanations have been discussed within an analysis team at the studied companies and the team agreed on the explanations given.

6.5.1 Practical Implications

No Difference in Lead-Times Between Phases: One explanation for the similarities of lead-times between phases is that large-scale system requires more specification and testing, and that system integration is more challenging when having systems of very large scale. Thus, systems documentation and management activities should only be removed with care in this context as otherwise there is a risk of breaking the consistency of the large system. Furthermore, there is no single phase that requires a specific focus on shortening the lead-time due to that there is no particularly time-consuming activity. Hence, in the studied context the result contradicts what would be expected from the assumptions made in literature. A consequence for practice is that one should investigate which are the value-adding activities in the development life-cycle, and reduce the non-value adding activities. An approach for that is lean software development [11].

No difference in Multi vs. Single System Requirements: The number of dependencies a requirement has does not increase the lead-time. An explanation is that with requirements affected by multiple systems the systems drive each other to be fast as they can only deliver value together. The same driving force is not found on single system requirements. However, we can hypothesize that single system lead-time can be shortened more easily, the reason being that waiting due to dependencies in a compound system requires the resolution of these dependencies to reduce the lead-time. On the other hand, no technical dependencies have to be resolved to remove lead-time in single systems.

Difference in Size: No major difference can be observed between small, medium and large requirements, except for large requirements in implementation and unit test. That is, at a specific size the lead-time for implementation and test increases drastically.

In consequence, there seems to be a limit that the size should have to avoid longer lead-times. This result is well in line with the findings presented in literature (cf [7, 1]).

6.5.2 Research Implications

The study investigated a very large system, hence research should focus on investigating time consumption for different contexts (e.g. small systems). This helps to understand the scalability of agile in relation to lead-times and with that the ability to respond quickly to market needs. Furthermore, the impact of size on lead-time is interesting to understand in order to right-size requirements.

In this study we have shown the absence of explanatory power for the variance in lead-time for phases and system impact. As time is such an important outcome variable research should focus on investigating the impact of other variables (e.g. experience, schedule pressure, team and organizational size, distribution, etc.) in a broader scale. Broader scale means that a sample of projects should be selected, e.g. by using publicly available repository with project data.

6.6 Conclusion

This paper evaluates software development lead-time in the context of a large-scale organization using incremental and agile practices. The following observations were made regarding lead-time:

- Phases do not explain much of the variance in lead-time. From literature one would expect that implementation and testing are the most time-consuming activities in agile development. However, due to the context (large-scale) other phases are equally time-consuming.
- There is no difference in lead-time for singe-system and multi-system requirements. This finding also contradicts literature. An explanation is that if a requirement has impact on multiple systems these systems drive each other in implementing the requirements quickly.
- With increasing size the lead-time within the implementation phase increases. This finding is in agreement with the related work.

In future work lead-times should be investigated in different contexts to provide further understanding of the behavior of lead-times in incremental and agile development.

6.7 References

[1] Manish Agrawal and Kaushal Chari. Software effort, quality, and cycle time: A study of cmm level 5 projects. *IEEE Trans. Software Eng.*, 33(3):145–156, 2007.

[2] Mikio Aoyama. Issues in software cycle time reduction. In *Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*, pages 302–309, 1995.

[3] Kent Beck. Embracing change with extreme programming. *IEEE Computer*, 32(10):70–77, 1999.

[4] Lars Bratthall, Per Runeson, K. Adelswärd, and W. Eriksson. A survey of lead-time challenges in the development and evolution of distributed real-time systems. *Information & Software Technology*, 42(13):947–958, 2000.

[5] Erran Carmel. Cycle time in packaged software firms. *Journal of Product Innovation Management*, 12(2):110–123, 1995.

[6] Bill Carreira. *Lean manufacturing that works: powerful tools for dramatically reducing waste and maximizing profits*. American Management Association, New York, 2005.

[7] Donald E. Harter, Mayuram S. Krishnan, and Sandra A. Slaughter. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science*, 46(4), 2000.

[8] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82(9), 2009.

[9] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404, 2010.

[10] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *Proceedings of the 10th International Conference on Product-Focused Software Process Improvement (PROFES 2009)*, pages 386–400, 2009.

[11] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, Boston, 2003.

[12] Melissa A. Schilling. Technological lockout: an integrative model of the economic and strategic factors driving technology success and failure. *Academy of Management Review*, 23(2):267–284, 1998.

[13] George Stalk. Time - the next source of competitive advantage. *Harvard Business Review*, 66(4), 1988.

[14] Glen L. Urban, Theresa Carter, Steven Gaskin, and Zofia Mucha. Market share rewards to pioneering brands: an empirical analysis and strategic implications. *Management Science*, 32(6):645–659, 1986.

[15] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in Software Engineering: An Introduction (International Series in Software Engineering)*. Springer, 2000.

Chapter 7

Software Process Improvement through the Lean Measurement (SPI-LEAM) Method

Kai Petersen and Claes Wohlin
Accepted for Publication in Journal of Systems and Software

7.1 Introduction

Software process improvement aims at making the software process more efficient and increasing product quality by continuous assessment and adjustment of the process. For this several process improvement frameworks have been proposed, including the Capability Maturity Model Integration (CMMI) [8] and the Quality Improvement Paradigm (QIP) [2, 4]. These are high level frameworks providing guidance what to do, but not how the actual implementation should look like. The Software Process Improvement through the Lean Measurement (SPI-LEAM) method integrates the software quality improvement paradigm with lean software development principles. That is, it describes a novel way of how to implement lean principles through measurement in order to initiate software process improvements.

The overall goal of lean development is to achieve a continuous and smooth flow of production with maximum flexibility and minimum waste in the process. All activities and work products that do not contribute to the customer value are considered waste.

Identifying and removing waste helps to focus more on the value creating activities [9, 34]. The idea of focusing on waste was initially implemented in the automotive domain at Toyota [33] identifying seven types of waste. The types of waste have been translated to software engineering into extra processes, extra features, partially done work (inventory), task switching, waiting, motion, and defects [31]. Partially done work (or inventory) is specifically critical [22]. The reason for inventory being a problem is not that software artifacts take a lot of space in stock, but:

- Inventory hides defects that are thus discovered late in the process [22].
- Time has been spent on artifacts in the inventory (e.g., reviewing of requirements) and due to change in the context the requirements become obsolete and thus the work done on them useless (see Chapter 3).
- Inventory impacts other wastes. For example, a high level of inventory causes waiting times. Formally this is the case in waterfall development as designers have to wait until the whole requirements document has been approved [30]. Long waiting times bare the risk of completed work to become obsolete. Furthermore, high inventory in requirements engineering can be due to that a high number of extra features have been defined.
- Inventory slows down the whole development process. Consider the example of a highway, if the highway is overloaded with cars then the traffic moves slowly.
- High inventory causes stress in the organization [25].

Lean manufacturing has drastically increased the efficiency of product development and the quality of products in manufacturing (see for example [9]). When implemented in software development lean led to similar effects (cf. [25, 23]). Even though lean principles are very promising for software development, the introduction of lean development is very hard to achieve as it requires a large shift in thinking about software processes. Therefore, an attempt to change the whole organization at once often leads to failure. This has been encountered when using lean in manufacturing [28] and software development [22].

To avoid the risk of failure when introducing lean our method helps the organization to arrive at a lean software process incrementally through continuous improvements. The method relies on the measurement of different inventories as well as the combined analysis of inventory measurements. The focus on inventory measurement is motivated by the problems caused by inventories discussed earlier. Furthermore, inventories also show the absence of lean practices and thus can be used as support when arguing for the introduction of the principles. In the analysis of the inventories a system thinking

method is proposed as lean thinking requires a holistic view to find the real cause of problems. That is, not only single parts of the development process are considered, but the impact of problems (or improvement initiatives) on the overall process have to be taken into consideration.

Initial feedback on SPI-LEAM was given from two software process improvement representatives at Ericsson AB (see [15]). The objective was to solicit early feedback on the main assumptions and steps of SPI-LEAM from the company, which needs triggered the development of the method.

The remainder of the paper is structured as follows: Section 7.2 presents the related work on lean software development in general and measurement for lean software development in particular. Section 7.3 presents the Software Process Improvement through the Lean Measurement (SPI-LEAM) Framework. Section 7.4 presents a preliminary evaluation of the method. Section 7.5 discusses the proposed method with focus on comparison to related work, practical implications, and research implications. Section 7.6 concludes the paper.

7.2 Related Work

7.2.1 Lean in Software Engineering

Middleton [22] conducted two industrial case studies on lean implementation in software engineering, and the research method used was action research. The company allocated resources of developers working in two different teams, one with experienced developers (case A) and one with less experienced developers (case B). The responses from the participants was that initially the work is frustrating as errors become visible almost immediately and are returned in the beginning. In the long run though the number of errors dropped dramatically. After the use of the lean method the teams were not able to sustain the lean method due to organizational hierarchy, traditional promotion patterns, and the fear of forcing errors into the open.

Another case study by Middleton et al. [23] studied a company practicing lean in their daily work for two years. They found that the company had many steps in the process not being value-adding activities. A survey among people in the company showed that the majority supports lean ideas and thinks they can be applied to software engineering. Only a minority (10 %) is not convinced of the benefits of lean software development. Statistics collected at the company show a 25 % gain in productivity, schedule slippage was reduced to 4 weeks from previously months or years, and time for defect fixing was reduced by 65 % - 80 %. The customer response on the product released using lean development was overwhelmingly positive.

Perera and Fernando [29] compared an agile process with a hybrid process of agile and lean in an experiment involving ten student projects. One half of the projects was used as a control group applying agile processes. A detailed description of how the processes differ and which practices are actually used was not been provided. The outcome is that the hybrid approach produces more lines of code and thus is more productive. Regarding quality, early in development more defects are discovered with the hybrid process, but the opposite trend can be found in later phases, which confirms the findings in [22].

Parnell-Klabo [27] followed the introduction of lean and documented lessons learned from the introduction. The major obstacles in moving from agile are to obtain open office space to locate teams together, gain executive support, and training and informing people to reduce resistance of change. After successfully changing with the help of training workshops and use of pilot projects positive results have been obtained. The lead-time for delivery has been decreased by 40 % - 50 %. Besides having training workshops and pilots sitting together in open office-landscapes and having good measures to quantify the benefits of improvements are key.

7.2.2 Lean Manufacturing and Lean Product Development

Lean principles initially focused on the manufacturing and production process and the elimination of waste within these processes that does not contribute to the creation of customer value. Morgan and Liker [24] point out that today competitive advantage cannot be achieved by lean manufacturing alone. In fact most automotive companies have implemented the lean manufacturing principles and the gap in performance between them is closing. In consequence lean needs to be extended to lean product development, not only focusing on the manufacturing/production process. This trend is referred to as lean product development which requires the integration of design, manufacturing, finance, human resource management, and purchasing for an overall product [24]. Results of lean product development are more interesting for software engineering than the pure manufacturing part as the success of software development highly depends on an integrative view as well (requirements, design and architecture, motivated teams, etc.), and at the same time has a strong product focus.

Morgan and Liker [24] identified that inventory is influenced by the following causes: batching (large hand-overs of, for example, requirements), process and arrival variation, and unsynchronized concurrent tasks. The causes also have a negative effect on other wastes: batching leads to overproduction; process and arrival variation leads to overproduction and waiting; and unsynchronized tasks lead to waiting. Thus, quantifying inventory aids in detecting the absence of lean principles and can be mapped to root causes. As Morgan and Liker [24] point out their list of causes is not complete.

Hence, it is important to identify the causes for waste after detecting it (e.g. in form of inventories piling up).

Karlsson and Ahlströhm [18] identified hinders and supporting factors when introducing lean production in a company in an industrial study. The major hinders are: (1) It is not easy to create a cross-functional focus as people feel loyal to their function; (2) Simultaneous engineering is challenging when coming from sequential work-processes; (3) There are difficulties in coordinating projects as people have problems understanding other work-disciplines; (4) It is challenging to manage the organization based on visions as people were used to detailed specifications and instructions; (5) The relationship to customers is challenging as cost estimations are expected, even in a highly flexible product development process. Factors helping the introduction of lean product development are: (1) Lean buffers in schedules; (2) Close cooperation with customers in product development to receive feedback; (3) For moving towards a new way of working successfully high competence engineers have to be involved; (4) Commitment and support from top management is required; (5) regular face-to-face meetings of managers from different disciplines.

Oppenheim [26] presents an extension of the value-stream mapping process to provide a comprehensive framework for the analysis of the development flow in lean product development. The framework is split into different steps. In the first step a takt-period is selected. A takt is a time-box in which different tasks are fulfilled and integrated. The second step is the creation of a current-state-map of the current process. The current-state map enables the identification of wastes and is the basis for the identification of improvements (e.g. the relations between waiting times and processing times become visible). Thereafter, the future-state-map is created which is an improved version of the map. Oppenheim stresses that all participants of the value-stream mapping process must agree to the future-state-map. After having agreed to the map the tasks of the map are parsed into the takt times defined in the first step. The last step is the assignment of teams (team architecture) to the different tasks in the value stream map. A number of success measures have been defined for the proposed approach: Amount of throughput time cut in comparison to competitors or similar completed programs; Amount of waste removed in the value-stream map (time-units or monetary value); Deviation of the planned value stream and the real value stream; Morale of the teams in form of a survey. Furthermore, the article points out that the goal of lean is to become better, cheaper, and faster. Though the reality often was that cheaper and faster was achieved on the expense of better (i.e. quality). One possible reason could be that no combined analysis of different measures was emphasized. Instead, measures were proposed as separate analysis tools (see Maskell and Baggaley [21] for analysis tools in the manufacturing context), but there is no holistic measurement approach combining individual measures to achieve a comprehensive analysis.

7.3 SPI-LEAM

The framework is based on the QIP [3] and consists of the steps shown in Figure 7.1. The steps according to Basili [3] are 1) Characterize the Current Project, 2) Set Quantifiable Goals and Measurements, 3) Choose Process Models and Methods, 4) Execute Processes and Collect and Validate Collected Data, 5) Analyze Collected Data and Recommend Improvements, and 6) Package and Store Experiences Made. The main contribution of this paper is to present a solution to step 2) for lean software development. The steps marked gray apply our method to achieve continuous improvement towards a lean software process.

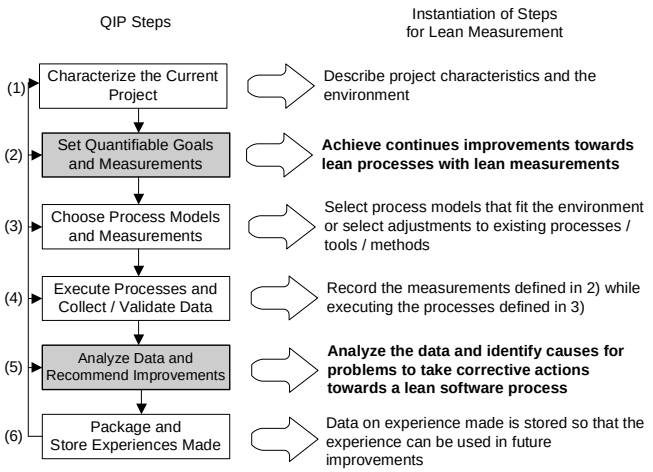


Figure 7.1: SPI-LEAM: Integration of QIP and Lean Principles

The steps are executed in an iterative manner so that one can always go back to previous steps and repeat them when needed. The non-expended steps are described on a high level and more details are provided for the steps expanded in relation to lean. The expansion of the second step (Set Quantifiable Goals and Measures) is explained in Section 7.3.2, and the expansion of the fifth step (Analyze Collected Data and Recommend Improvements) in Section 7.3.3.

1. *Characterize the Current Project:* In the first step, the project characteristics and the environment are characterized. This includes characteristics such as application domain, process experience, process expertise, and problem constraints [3].

2. *Set Quantifiable Goals and Measurements:* It is recommended to use the goal-question-metric approach [2] to arrive at the goals and associated measurement. The goal to be achieved with our method is to enable continuous software process improvement leading to a lean software process. The emphasis is on achieving continuous improvement towards lean software processes in order to avoid the problems when introducing an overall development paradigm and getting acceptance for the change, as seen in [22, 28]. The goals of achieving a lean process are set by the researchers to make the purpose of the method clear. When applying the method in industry it is important that the goals of the method are communicated to industry. In order to achieve the goals two key questions are to be answered, namely 1) what is the performance of the development process in terms of inventories, and 2) what is the cause of performance problems? The answer to the first question should capture poor performance in terms of different inventories. Identifying high inventory levels will help to initiate improvements with the aim of reducing the inventories and by that avoiding the problems caused by them (see Section 7.1). The second question aims at identifying why the inventory levels are high. This is important to initiate the right improvement. To identify the inventories in the development process one should walk through the development process used in the company (e.g,. using value stream mapping aiming at identifying waste from the customers' perspective [31]). This is an important step as not all companies have the same inventories. For example, the company studied requires an inventory for product customizations. For a very generic process, as shown in Figure 7.2, we can find the following inventories: requirements, test cases, change requests, and faults. The inventories for faults and fault-slip-through represent the quality dimension. A detailed account and discussion of the inventories is provided in Section 7.3.2. The collection mechanism is highly dependent on the company in which the method is applied. For example, the collection of requirements inventory depends on the requirements tool used at the specific company.

3. *Choose Process Models and Methods:* This step decides which process model (e.g., waterfall approach, Extreme Programming (XP), SCRUM) to use based on the characteristics of projects and environment (Step 1). If, for example, the environment is characterized as highly dynamic an agile model should be chosen. We would like to point out that the process model is not always chosen from scratch, but that there are established models in companies that need further improvement. Thus, the choice can also be a modification to the process models or methods and tools used to support the processes. For example, in order to establish a more lean process one approach could be to break down work in

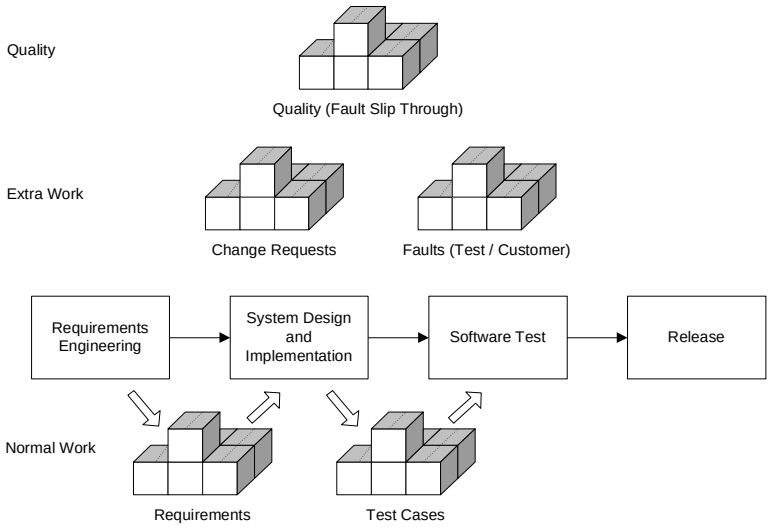


Figure 7.2: Inventories in the Software Process

smaller chunks to get things faster through the development process.

4. *Execute Processes and Collect and Validate Collected Data:* When the process is executed measurements are recorded. In the case of SPI-LEAM the inventory levels have to be continuously monitored. Thereafter, the collected data has to be validated to make sure that it is complete and accurate.

5. *Analyze Collected Data and Recommend Improvements:* In order to improve performance in terms of inventory levels the causes for high inventories have to be identified. For understanding a specific situation and to evaluate improvement actions system thinking methods have been proven to be successful in different domains. Having found the cause for high inventory levels, a change to the process or the methods used has to be made.

6. *Package and Store Experiences Made:* In the last step the experience made has to be packaged, and the data collected needs to be stored so the experiences made are not lost and can be used for future improvements.

The following section provides an overview of how SPI-LEAM aids in continuously improving software processes to become more lean. This includes detailed

descriptions of the expansions in steps two and five of the QIP.

7.3.1 Lean Measurement Method at a Glance

The first part of the method is concerned with setting quantifiable goals and measurements (Second step in the QIP). This includes the measurement of individual inventories. Thereafter, the measures of individual inventories are combined with each other, and with quality measurements (see Figure 7.3). How to measure individual inventories and combine their measurements with quality are explained in Section 7.3.2. For analysis purposes it is recommended not to use more than five inventories to make the analysis manageable. It is also important that at least one inventory focuses on the quality dimension (faults, fault-slip-through). Each measure should be classified in two different states, namely high inventory levels and low inventory levels. With that restriction in mind the company can be in $2^5 = 32$ different states (2 corresponding to high and low inventory levels, and 5 corresponding to the number of measurements taken). However, as the technique allows to measure on different abstraction levels, one individual inventory level (e.g. requirements) is derived from several sub-inventories (e.g. high level requirements, detailed requirements). Thus, we believe that companies should easily manage with a maximum of five inventories without neglecting essential inventories. Table 7.1 summarizes the goals, questions, and metrics according to the Goal-Question-Metric approach [2]. The inventories in the table are based on activities identified in the software engineering process (see e.g. [17]). As mentioned earlier each company should select the inventories relevant to their software processes.

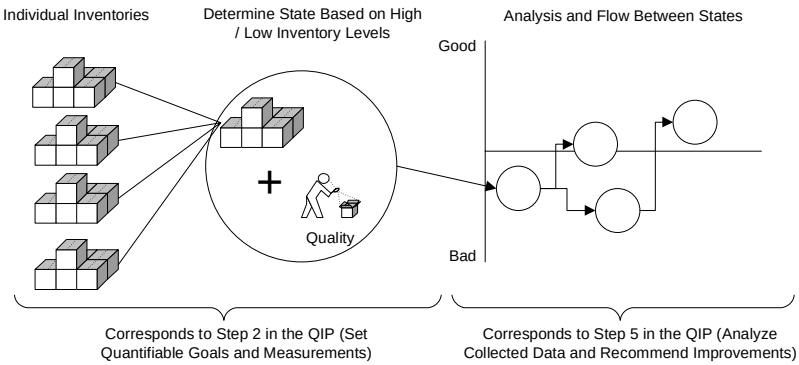


Figure 7.3: Method at a Glance

The second part of the method is concerned with the analysis of the situation, i.e.

Table 7.1: Goal Question Metric for SPI-LEAM

Dimension	Specification
Goals	• Enable continuous software process improvement leading to a lean software process. • Avoid problems related to resistance of change by improving in a continuous manner.
Questions	• Q1: What is the performance of the development process in terms of inventories? • Q2: What is the cause of performance problems?
Metrics	• Requirements (Individual Inv.) – High level Req. (Sub.-Inv.) – Detailed Req. – Req. in Design and Impl. – Req. in Test • Test Cases (Individual Inv.) – Unit Test (Sub.-Inv.) – Function Test – Integration Test – System Test – Acceptance Test • Change Requests (Individual Inv.) – CR under Review – Approved CRs – CRs ready for Impact Analysis – CRs in Test • Faults and Failures (Individual Inv.) – Internal Faults and Failures (Test) – External Faults and Failures (Customer) • Fault Slip Through (Quality) – Req. Review Slippage – Unit Test Slippage – Function Test Slippage – etc.

the aim is to determine the causes for high inventory level and quality problems. In other words, we want to know why we are in a certain state. Based on the analysis it is determined to which state one should move next. For example, which inventory level should be reduced first, considering the knowledge gained through the analysis. To make the right decisions the end-to-end process and the interaction of its parts are taken into consideration. As system theory in general and simulation as a sub-set of system theory methods allows to consider the interactions between individual parts of the process (as well as the process context) we elaborate on which methods are most suitable to use in Section 7.3.3.

7.3.2 Measure and Analyze Inventory Levels

The measurement and analysis takes place in the second step of the QIP as shown in the previous section. The measurement and analysis of inventory requires two actions:

- *Action 1:* Measure the individual inventories, i.e. change requests, faults, requirements, and test cases (see Figure 7.2). Each individual inventory (e.g., requirements) can have sub-inventory (e.g. high level requirements, detailed requirements, requirements in implementation, requirements in test) and thus these inventories are combined to a single inventory "requirements". The measurement of the inventories is done continuously to be able to understand the change of inventories over time. Inventories can also be split into work in process and buffer. The advantage is that this allows to determine waiting times, but it also requires to keep track of more data points.
- *Action 2:* Determine the overall state of the software process for a specific product combining the individual inventory levels. The combination of individual inventories is important as this avoids unintended optimization of measurements. That is, looking on one single inventory like requirements can lead to optimizations such as skipping requirements reviews in order to push the requirements into the next phase quicker. However, when combining inventory measures with each other and considering inventories representing the quality dimension will prevent this behavior. As a quality dimension we propose to use fault-slip through measurement [10] and the fault inventory.

Step 1: Measure Individual Inventories

Measurement Scale: In this step each of the inventories is measured individually considering the effort required for each item in the inventory. In manufacturing inventory is counted. In software development just counting the inventory is insufficient as artifacts

in software development are too different. For example, requirements are of different types (functional requirement, quality requirement) and of different complexities and abstraction levels (see for example [15]). Furthermore, requirements might never leave the inventory as they are always valid (e.g. requirements on system response times). Similar issues can be found for testing, like test cases that are automated for regression testing are different from manual test cases. As a consequence we propose to take into account the effort required to implement a requirement, or to run a test case. To make the importance of considering effort more explicit, take the example of a highway again. It is a difference if we put a number of trucks on a crowded highway, or a number of cars. In terms of requirements this means that a high effort requirement takes much more space in the development flow than small requirements. For each of the inventories there are methods available to estimate the effort:

- *Requirements:* Function points have been widely used to measure the size of systems in terms of requirements. This can be used as an input to determine the effort for requirements [7]. Another alternative is to estimate in intervals by classifying requirements as small, medium, or large. For each class of requirements an interval has to be set (e.g. in person days).

- *Test Cases:* Jones [17] proposed a model to estimate the effort of testing based on function point analysis. Another approach combines the number of test cases, test execution complexity, and knowledge of tester to estimate the effort of test execution [35].

- *Change Requests:* The effort of a change is highly influenced by the impact the change has on the system. Therefore, impact analysis can be used to provide an indication of how much effort the change requires [19].

- *Faults:* A method for corrective maintenance (e.g. fixing faults) has been evaluated by De Lucia et al. [20] considering the number of tasks needed for maintenance and the complexity of the system.

Describe Individual Inventory Levels: Individual inventories are broken down further into sub-inventories when needed. For example, requirements can be broken down into high level requirements, detailed requirements, requirements in design phase, and requirements in release. Similar, test cases can be broken down into unit test cases, integration test cases, system test cases, etc. The measurement of effort in inventory is done on the lowest level (i.e. high level requirements inventory, detailed requirements inventory, and so forth) and later combined to one single measure for requirements inventory. In Figure 7.4 the example for requirements (fictional data) and related sub-inventories

is illustrated using a radar chart. We use this way of illustrating inventories throughout the whole analysis as it allows to see the distribution of effort between inventories in one single view very well. Based on this view it is to be determined whether the inventory level is high or low for the specific company. As mentioned earlier this depends on the capacity available to the company. Therefore, for each dimension one has to compare the capacity with the inventory levels. The situation in Figure 7.4 would suggest that there are clear overload situations for high level requirements and requirements in implementation. The inventory of requirements to be specified in detail is operating at its maximum capacity while there is free capacity for requirements in test.

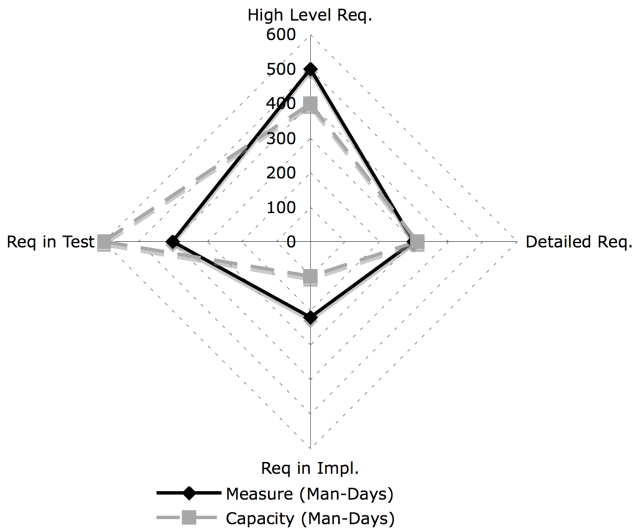


Figure 7.4: Measuring Requirements Inventory

Simulating Overload Situations: One should aim for a situation where the process is not overloaded, but the process should be stressed as much as possible. Figure 7.5 illustrates the situation of overload. If the load (inventory) is higher than the capacity then there is a high overload situation (in analogy to the highway there are more cars on the road than the road was designed for) and thus the flow in the process is jammed. A critical overload situation means close to complete standstill. If the capacity is almost completely utilized (the highway is quite full) then the flow moves slower, but still has a steady flow. Below that level (good capacity utilization) the resources are just used right, while when reducing the load too much an underload situation is created. To

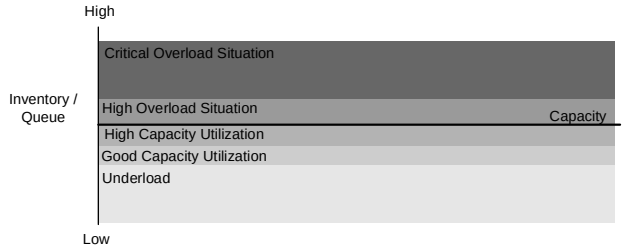


Figure 7.5: Good and Bad Inventory Levels

determine which are the thresholds for critical, high, and overload situations, queuing theory can be used. The inventory (e.g. requirements) represents the queue and the activity (e.g. system design and implementation) represents the server. An example of such a simulation for only requirements can be found in Höst et al. [16].

Set Inventory Level to High or Low: With the knowledge of when the organization is in an overload situation one can measure whether the inventory level is high or low. As we only allow two values in the further analysis (i.e. high and low) for reducing the complexity of the analysis we propose to classify inventories within the areas ”good capacity utilization” and ”underload” as low, and the ones above that as high. In order to combine the different sub-inventories for requirements into one inventory different approaches are possible, like:

- After having simulated the queues and knowing the zones the overall inventory level is high if the majority of sub-inventories is rated high.
- Sub-inventories are not independent (e.g. the queue for detailed requirements is influenced by the queue of high level requirements and the capacity of the server processing the high level requirements). Thus, more complex queuing networks might be used to determine the critical values for high and low.
- We calculate the individual inventory level II as the sum of the difference between capacity of the server for inventory i (C_i) and the actual measure for inventory i (A_i), divided by the number of sub-inventories n.

$$II = \frac{\sum_i^n (C_i - A_i)}{n} \tag{7.1}$$

If the value is negative then on average the company operates above their capacity. Thus, one should strive for a positive value which should not be too high as

this would mean that the company operates with underload.

Determine the State of the Process Combining Inventories and Quality Measurement

As mentioned earlier it is good to restrict the number of states (e.g. to $2^5 = 32$) as this eases the analysis and it is possible to walk through and analyze the possible states in more depth. If this restriction is not feasible the number of states grows quite quickly, and with that the complexity of analysis. In order to illustrate the combined inventory level we propose to draw a spider web again with the average efforts for sub-inventories, the average capacity for each individual inventory and the rating as high or low. Even though we know the state for each inventory level the reason for drawing the diagram is that it allows to see critical deviations between inventories and capacity, as well as effort distributions between inventories. Besides the inventories it is important to consider the quality dimension. The reason is that this avoids an unintended optimization of measures. We propose to use the fault slip through (FST) and the number of faults to represent the quality dimension. The number of faults is a quality indicator for the product while the FST measures the efficiency of testing.

The FST measure helps to make sure that the right faults are found in the right phase. Therefore, a company sets up a test strategy determining which type of fault can (and should) be found in which test phase (e.g. function test). If the fault is found later (e.g. system test instead of function test) then the fault is considered a slip-through. Experience has shown that the measure in Equation 7.2 is one of the most useful [11]:

$$PIQ = \frac{SF}{PF} \tag{7.2}$$

The measure in Equation 7.2 shows the Phase Input Quality to a specific phase (phase X). *SF* measures the number of faults that are found in phase X, but should have been found earlier. *PF* measures the total number of faults found in phase X. When conducting this measurement it is also important to consider whether overall a high or low number of faults are found in phase X. In order to determine whether the testing strategy is in a good or bad state consider Figure 7.6. The figure shows four states based on 1) whether the overall number of faults found in phase X can be considered as high or low, and 2) whether the PIQ is high or low. Combining these dimensions leads to a high or low FST-figure. Whether the number of faults in phase X is high is context dependent and can, for example, be determined by comparing the number of faults found across different test phases.

The four dimensions in Figure 7.6 can be characterized as follows:

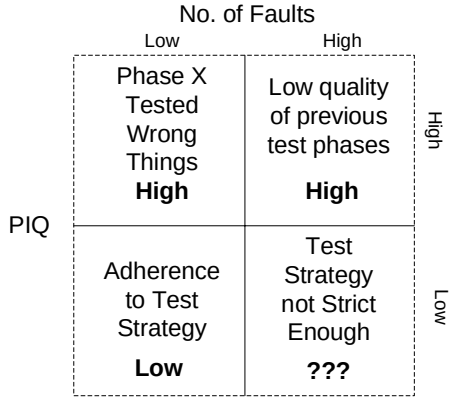


Figure 7.6: FST-Level

- *(No. Faults High, PIQ High)*: The fault-slip of faults to phase X is quite high, as well as the overall number of faults is high. This is an indicator for quality issues and low testing efficiency. We assign the value high to the FST measure.
- *(No. Faults High, PIQ Low)*: In this situation the test strategy is not strict enough and should probably require that more faults should be found in earlier phases. As the data is based on a flawed test strategy the results should not be used as an indicator for process performance.
- *(No. Faults Low, PIQ High)*: Phase X probably tested the wrong things as one can assume that more faults are discovered considering a high fault-slip in earlier phases. We assign the value high to the FST measure.
- *(No. Faults Low, PIQ Low)*: The process adheres to the testing strategy and few faults are discovered which is an indicator of good quality. We assign the value low to the FST measure.

As effort is used throughout the method it is important to mention that FST can be transfered into effort as well. With the knowledge of average effort for fixing a fault found in a certain phase the improvement opportunity can be calculated.

In summary, the result of the phase are:

- A radar chart showing the average efforts and capacities related to each individual inventory, and the result of the improvement opportunity in terms of effort for faults found late in the process (FST).

- A description of the values of the inventory levels rated as either low or high with at least one inventory representing the quality dimension.

7.3.3 Analysis and Flow Between States

The analysis focuses on understanding the reasons for the current situation and finding the best improvement alternatives to arrive at an improved state. As a simple example consider a situation with one inventory (test cases) and the FST measure (see Figure 7.7). Analyzing the situation with just two measures shows that 1) no inventory measures should be taken without quality, and 2) combining measures allows a more sophisticated analysis than looking at different measures in isolation. Consider the situation in Figure 7.7 with one inventory (Test Cases) and the FST measure, both being labeled as either high or low based on the analysis presented before. Four different states are possible:

1. *(TC high, FST high)*: In this situation the company is probably in a critical situation as they put high effort in testing and at the same time testing is inefficient. Thus, this situation means that one has to explore the best actions for improvements. That is, one has to consider why the testing process and testing techniques lead to insufficient results. Possible sources might be found in the context of the process (e.g. requirements which form the basis for testing) or problems within the process (e.g. competence of testers).

2. *(TC high, FST low)*: This state implies that the company delivers good quality and puts much effort to do so in terms of test cases. Therefore, one does not want to move away from the low FST status, but wants to improve testing efficiency in order to arrive at the same result for FST without loosing quality. An option for improvement could be to switch to more automated testing so the effort per test case is reduced.

3. *(TC low, FST high)*: The test effort in terms of test cases is good, but there is low efficiency in testing. Either too little effort is spent on testing or the testing process and test methods need to be improved.

4. *(TC low, FST low)*: The state is good as testing is done in an efficient way with a low value for FST.

The analysis of the situation makes clear that it is important to include inventories representing the quality of the software product. Situations 1 and 2 are the same in terms of test case level, but lead to different implications when combined with the FST measure.

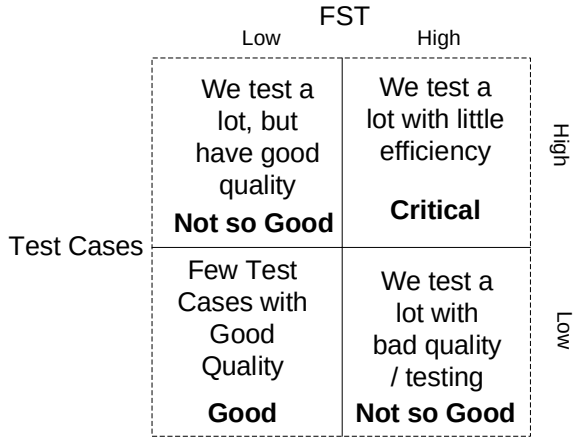


Figure 7.7: Analysis with Test Cases and FST

Analysis with n Inventories

With only two measures this seems to be obvious. Though, the outcome of the analysis will change when adding more inventories, and at the same time the analysis becomes more challenging. In order to characterize the situation of the company in terms of inventory levels the state of the company has to be determined. The state is defined as s tuple S of inventories s_i:

$$S := (s_1, s_2, s_3, ..., s_n),\ s_i \in \{high, low\} \tag{7.3}$$

What we are interested in is how improvement actions lead to a better state in terms of inventories. An ideal state would be one where all inventories have a good capacity utilization as defined in Figure 7.5. From an analysis point of view (i.e. when interpreting and predicting the behavior of development based on improvement actions) we assume that only one inventory changes at a time. When analyzing how to improve the state of inventories alternative improvement actions need to be evaluated. That is, the company should aim at reaching the desired state by finding the shortest path through the graph, the reason being to reduce the time of improvement impact. Figure 7.8 shows an example of state changes to achieve a desired state illustrated as a directed graph. The solid and dashed lines represent two different decisions regarding improvement actions. In the case of the graph decision one would be preferable to decision two as the desired state is achieved in fewer state changes, assuming the edges all have the

same value.

Methods that can be used to support decision makers in the organization to make this analysis are presented in Section 7.3.3. In order to make the theoretical concepts in the case of n inventories more tangible the following section presents an application scenario.

Application Scenario for Analysis with n States

The following inventories are considered in the application scenario and thus represent the state of the company S:

$$S := (Requirements, Change\ Requests, Faults, Test\ Cases, Fault\ Slip) \tag{7.4}$$

In the start state testing was done on one release and correct defects were found according to the test strategy. Furthermore, the release can be characterized as mature and thus it is stable, resulting in a low number of change requests. Though, the number of requirements is high as a project for the next release will be started. In addition to that testing has identified a number of faults in the product that need to be resolved. This leads to the following start state:

$$S_0 := (high, low, high, low, low) \tag{7.5}$$

The decision taken to achieve a more lean process in terms of inventory levels is to put additional resources on fixing the faults found in testing and wait with the new project until faults are resolved. Adding resources in testing leads to a reduction in the number of faults in the system.

$$S_1 := (high, low, low, low, low) \tag{7.6}$$

Due to fault fixing regression testing becomes necessary, which increases the number of test-cases to be run, leading to state S_2.

$$S_2 := (high, low, low, high, low) \tag{7.7}$$

Now that testing is handled the work starts on new requirements which leads to a reduction in requirements inventory. Furthermore, the value for FST could change depending on the quality of the tests conducted. This leads to a new current state:

$$S_3 := (low, low, low, high, low) \tag{7.8}$$

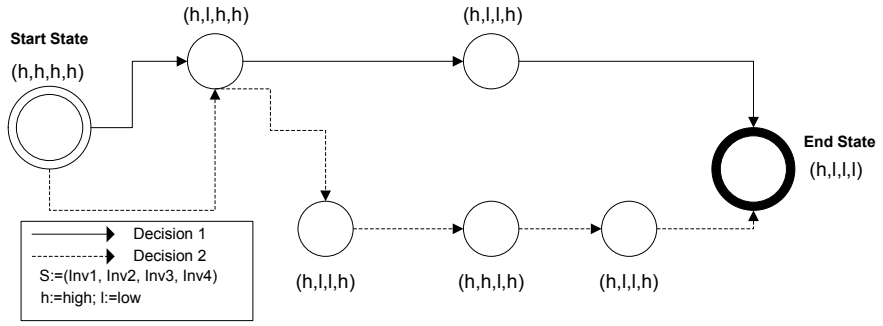


Figure 7.8: Improving the State of the Inventories

Analysis Support

Reasoning with different inventories to take the right actions is supported by the lean principle of "see the whole". In other words, a combined analysis of different parts of a whole (and the interaction between the parts) have already been considered when combining inventories and quality. As a solution for handling the analysis and evaluation of improvement actions different solutions are available which need to be evaluated and compared for their suitability in the lean context. Systems thinking as a method has been proven successful to conduct complex analyses. Three type of system approaches are common, namely hard systems, soft systems, and evolutionary systems.

- *Hard Systems:* These kind of systems are used for a quantitative analysis not considering soft factors [13]. The systems are usually described in quantitative models, such as simulation models. Candidates for analyzing the above problem are continuous simulations combined with queuing theory [12], or Discrete Event Simulations with queuing theory [16]. When repeating the activity in a continuous matter a major requirement on the model is to be simple and easily adjustable, but accurate enough. Fulfilling this requirement is the challenge for future research in this context.

- *Soft Systems:* Soft systems cannot be easily quantified and and contain interactions between qualitative aspects such as motivations, social interactions etc. Those problems are hard to capture in quantitative simulations and therefore some problems will only be discovered using soft system methodologies [13]. Therefore, they might be used as a complement to hard systems. In order to

visualize and understand soft systems, one could make use of mind-maps or scenarios and discuss them during a workshop.

- *Evolutionary Systems:* This type of system applies to complex social systems that are able to evolve over time. However, those systems are very specific for social systems of individuals acting independently and are therefore not best suited from a process and workflow perspective.

After having decided on an improvement alternative the action is implemented and the improvements are stored and packaged. Thereby, it is important not just to describe the action, but take the lessons learned from the overall analysis as this will provide valuable data of behavior of the process in different improvement scenarios.

7.4 Evaluation

7.4.1 Static Validation and Implementation

The purpose of the static validation is to get early feedback from practitioners regarding improvements and hinders in implementing the proposed approach. Another reason for presenting the approach is to get a buy-in from the company to implement the approach [14]. In this case the method has been presented and discussed with two representatives from Ericsson AB in Sweden, responsible for software process improvement at the company. The representatives have been selected as they are responsible for identifying improvement potential in the company's processes, as well as to make improvement suggestions. Thus, they are the main stakeholders of such a method. The goal was to receive early feedback on the cornerstones of the methodology (e.g. the goals of the method; keeping work-load below capacity; combining different measurement dimensions; easy to understand representation of data), as well on limitations of the method. The following feedback was given:

- The practitioners agree with the observation that the work-load should be below the capacity. Being below capacity is good as this, according to the practitioners experience, makes the development flow more steady. Furthermore, a lower capacity situation provides flexibility to fix problems in already released software products or in handling customization requests.

- When introducing inventory measures at the company the issue of optimization of measures was considered early on in the approach. For example, in order to reduce the level of inventory in requirements one could quickly write requirements

and hand them over to the next phase to achieve measurement goals. In consequence, the company decided to consider inventories representing normal work (requirements flow) as well as quality related inventories (number of faults, and fault-slip-through). Furthermore, the company measures the number of requests from customers to provide individual customizations to their systems, which is an inventory representing extra work.

- The illustration of the data (capacity vs. load) in the form of radar charts was perceived as easy to understand by the practitioners, due to the advantages mentioned earlier. That is, one can gain an insight of several inventories at the same time, and observe how significant the deviation between capacity and load is.

The following limitation was observed by the practitioners: The method relies on knowing the capacity of the development in comparison to the work-load. The practitioners were worried that the capacity and work-load are hard to determine and are not comparable. For example, there is a high variance in developer productivity. Even though a team has N developers working X hours a day the real capacity is not equal to N*X hours. Furthermore, the work-load needs to be estimated as, for example, in terms of size of requirements. An estimation error would lead to an invalid analysis.

In summary, the practitioners perceived the method as useful in achieving a more lean software process. Based on the static validation and further discussions with the practitioners an implementation plan for the method was created (see Figure 7.9).

The first two steps are related to requirements inventories. The rational for starting with requirements was that implemented requirements provide value to the customer and thus are of highest priority in the implementation of SPI-LEAM.

The first step of the implementation plan is the creation of a web-platform to capture the requirements flow. The platform lists the requirements in each phase and allows the practitioners to move requirements between phases. When a requirement is moved from one phase to another the date of the movement is registered. Thus, the inventory level at any point in time for each phase of development can be determined.

The second step is the analysis of the requirements level by measuring the number of requirements in different phases. Complementarity to the inventory levels the data is also the basis to conduct analysis to get further insights into how requirements evolve during the development life-cycle. Cumulative flow diagrams were used to visualize the flow and throughput of development. In addition, the time requirements stayed in different phases were illustrated in the form of box-plots. The first and second steps have been completed and preliminary data collected in these step is presented in Section 7.4.2.

In the third step the number of faults is measured in a similar manner as the requirements flow. The company captures the flow of fixing the faults (e.g. number of

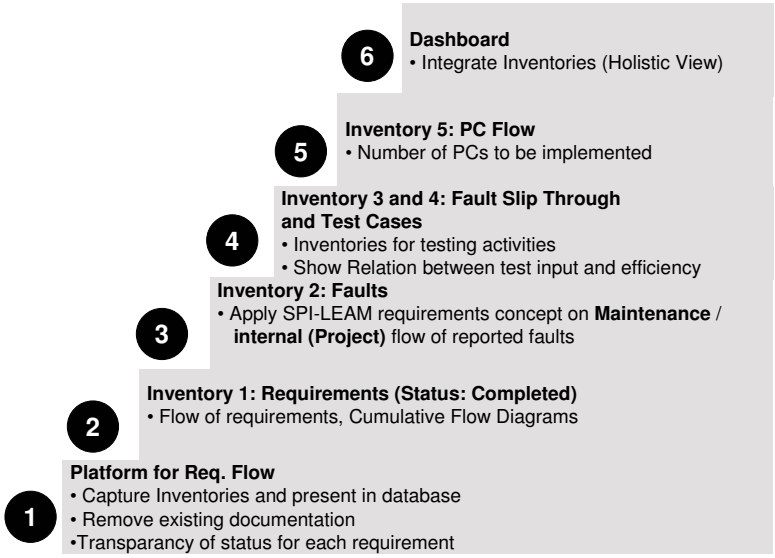


Figure 7.9: Implementation Steps of SPI-LEAM

faults registered, in analysis, implementation, testing, and regression testing). With this information one can calculate the number of requirements in different phases and construct cumulative flow diagrams as well. The flow of fixing faults is separated between internal (faults discovered by tests run by the company) and external (faults reported by the customer).

The fourth step in the staircase is concerned with measuring the number of test cases and the fault-slip through. An analysis for a combination of these inventories is shown in Figure 7.7.

The fifth step measures the number customization requests by customers. The development of customization follows an own flow which can be analyzed in-depth in a similar fashion as the flow of requirements (Step 2) and faults (Step 3).

In the last (sixth) step a dashboard is created which integrates the analysis of the individual inventory measurements on a high level (i.e. capacity vs. actual level of inventory) in form of a radar chart. To conduct an in-depth analysis a drill-down is possible. For example, if the inventory for requirements is high then one can investigate the requirements flow in more detail.

7.4.2 Preliminary Data

The requirements inventory was measured for a large-scale telecommunication product developed at Ericsson AB. The product was written in C++ and Java and consisted of over five million lines of code (LOC). Figure 7.10 shows an individual and moving range (I-MR) chart for the inventory of requirements in implementation (design, coding, and testing) over time. Due to confidentiality reasons no actual values can be reported. The continuous line in the figure shows the mean value while the dashed lines show the lower and upper control limits. The upper and lower control limits are $+/-$ two or three standard deviations away from the mean. A value that is outside the control limits indicates an instability of the process. The graph on the top of Figure 7.10 shows the individual values, while the graph on the bottom shows the moving range. The moving range is calculated as $MR = |X_i - X_{i-1}|$, i.e. it shows the size of the change in X between data point i and data point $i-1$.

The figure for the individual values shows a large area of data points outside the upper control limit. In this time period the data indicates a very high inventory level. When presenting the inventory data to a development unit the participants of the unit confirmed that development was fully utilized and people felt overloaded. This also meant that no other activity (e.g. refactoring) did take place besides the main product development. The opposite can be observed on the right-hand side of the figure where the data points are below the control limit. In this situation most of the requirements passed testing and were ready for release, which meant that the development teams idled from a main product development perspective. This time was used to solely concentrate on activities such as refactoring. To determine the capacity of development (and with that acceptable inventory levels) we believe it is a good and practical approach to visualize the inventories and discuss at which times the development teams felt overloaded or underloaded.

It is also interesting to look at the figure showing the moving range. A large difference between two data-points is an indication for batch-behavior, meaning that many requirements are handed over at once. Large hand-overs also constitute a risk of an overload-situation. From a lean perspective one should aim at having a continuous and steady flow of requirements into the development, and at the same time delivering tested requirements continuously.

Collecting inventory data in a continuous manner also enables the use of other lean analysis tools, such as cumulative flow diagrams [32, 1]. They allow to analyze the requirements flow in more detail with respect to throughput and cycle times. The graph in Figure 7.11 shows a cumulative flow diagram which is based on the same data as the control charts. The top line represents the total number of requirements in development. The line below that shows the number of requirements that have been detailed

and handed over to implementation. The next line shows the number of requirements handed over to node test, and so forth. The difference between two lines at a certain point in time shows the inventory. The lines themselves represent hand-overs.

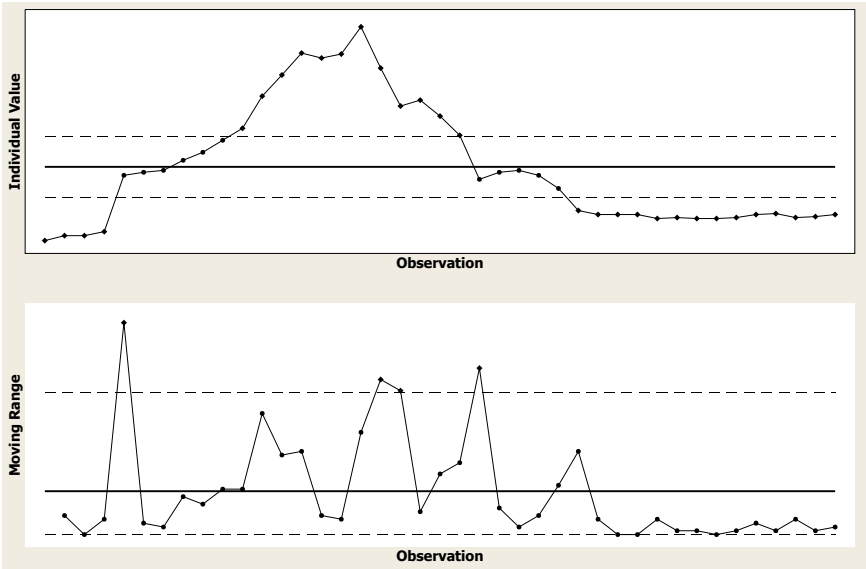


Figure 7.10: Inventory of Requirements in Implementation over Time (Observation = Time)

Cumulative flow diagrams provides further information about the development flow that can be used complementary to the analysis of inventory levels shown in Figure 7.10. The cumulative flow diagram shows velocity/throughput (i.e. the number of requirements handed over per time unit). Additionally it provides information about lead-times (e.g. the majority of the requirements is handed over to the release after 2/3 of the overall development time has passed). The batch behavior from the moving range graph can also be found in the cumulative flow diagram (e.g. large hand-over from node-test to system test in the middle of the time-line). A detailed account of measures related to development flow and the use of cumulative flow diagrams in the software engineering context can be found in Chapter 8.

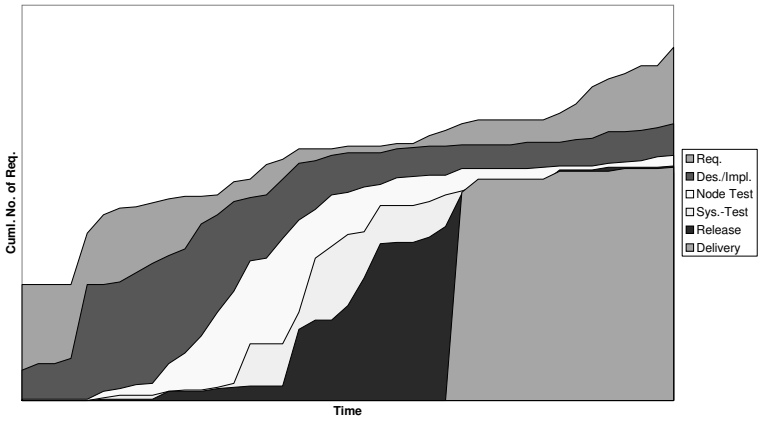


Figure 7.11: Cumulative Flow Diagram

7.4.3 Improvements Towards Lean

Based on the observations from measuring the requirements flow the company suggested a number of improvements to arrive at a more lean process. Two examples are provided in the following:

- *From Push to Pull:* A reason for the overload situation is that the requirements are pushed into development by allocating time-slots far ahead. Due to the analyses the desire is to change to a pull-approach. This should be realized by creating a buffer of prioritized requirements ready for implementation from which the development teams can pull. Thereby, the teams get more control of the work-load which is expected to help them in delivering in a more continuous manner.

- *Intermediate Release Versions:* Before having the final delivery of the software system with agreed scope intermediate releases should be established that that have the quality to be potentially released to the customer. Thereby the company wants to achieve to 1) have high quality early in development, and 2) have a motivator to test and integrate earlier.

It is important to emphasize that only a part of SPI-LEAM has been implemented so far. A more holistic analysis will be possible when having the other inventory measurements available as well. One interesting analysis would be to plot all inventories in graphs such as the ones presented in Figures 7.10 and 7.11 and have a look at them together. For example, an overload in the requirements flow would explain a slow-down

in the maintenance activities.

7.5 Discussion

7.5.1 Comparison with Related Work

A comparison with the related work allows to make two observations which indicate the usefulness of SPI-LEAM as an extension over existing measurements in the lean context.

First, Morgan and Liker [24] identified a mapping between root causes and waste. An interesting implication is that the occurrence of waste in fact points to the absence of lean practices. In consequence SPI-LEAM is a good starting point to identify wastes in development continuously. Based on the results provided by SPI-LEAM an undesired behavior of the development becomes visible and can be further investigated with techniques such as root cause analysis. This can be used as a motivation for management to focus on the implementation of lean practices. The transparency of the actual situation can also provide insights helping to overcome hindering factors, such as the ones reported in [18].

Secondly, Oppenheim [26] pointed out that it is important to not only focus on speeding up the development to achieve better cycle times for the expense of quality. To reduce this risk we proposed to provide inventory measurements that can be used to evaluate cycle times and throughput (see cumulative flow diagrams) and inventories related to quality together (flow of fixing faults). If, for example, cycle time is shortened by reducing testing this will show positively in the cycle time, but negatively in the fault-slip-through and the number of faults reported by the customers.

7.5.2 Practical Implications

The flexibility of the method makes it usable in different contexts. That is, the method allows companies to choose inventories that are relevant for their specific needs. Though, the only restriction is that at least one inventories should be considered that represents the quality dimension. The risk of optimizing measures on the cost of quality is thereby reduced. In addition, we presented different alternatives of analysis approaches (hard systems, soft systems, evolutionary systems) that companies can use to predict the effect of improvement actions.

Criticism was raised regarding the determination of capacity/work-load by the practitioners. In order to determine the right work-load, simulation can be used, in particular queuing networks and discrete event simulation (a theoretical description can be

found in [5, 6]). An example of the application of discrete event simulation with queuing networks to determine bottlenecks in the software engineering context is shown in [16]. As mentioned earlier, there are also alternative approaches that can be used which are easier and faster to realize (see description of soft systems in Section 7.3.3). Another approach is to plot inventory levels based on historical data and have a dialog with development teams and managers about the workload situation over time.

The practitioners also perceived the analysis and prediction of the effect of improvement actions (as shown in Figure 7.8) as theoretical, making them feel that the method might not be practical. However, this was due to the way of illustrating the movement between states as a directed graph. In practice, one would provide a view of the inventory levels over time illustrating them as a control chart/cumulative flow diagram as shown in Section 7.4.2. Doing so allowed the practitioners to identify a departure from lean practices. For example, the cumulative flow diagram derived from the inventory data showed that the system was not built in small and continuous increments. This observation helped to raise the need for a more continuous flow of development and led the company to take actions accordingly (see Section 7.4.3). In other words, SPI-LEAM provides facts that allow a stronger position when arguing why specific practices should receive more attention.

Considering the feedback from industry on the method it seems a promising new approach to continuously improve software processes to become more lean. As the method is an instantiation of the QIP one can leverage of the benefits connected to that paradigm (e.g. having a learning organization due to keeping track of the experiences made, see last step in Figure 7.1).

7.5.3 Research Implications

The related work shows that lean software engineering has been evaluated as a whole, i.e. it is not clear which tools have been applied. Furthermore, the benefit of single tools from the lean tool-box have not been evaluated so far to learn about their benefits in software engineering contexts. Such applications also show how the methods have to be tailored (see, for example, capacity discussion in Section 7.5.2). In general there are only few studies in software engineering and more studies are needed which describe the context in detail in which lean was applied, as well as how lean was implemented in the companies.

Concerning SPI-LEAM, there is a need to evaluate what kind of improvement decisions are proposed and implemented based on the measurement results derived by the method. This has to be followed up in the long run to see whether continuous improvements will be achieved in terms of making the software process more lean. In addition, SPI-LEAM has to be applied in different contexts to learn what kind of inventories

are the most relevant in specific contexts. For example, how does SPI-LEAM differ between different agile processes (SCRUM or eXtreme Programming) and different complexities (large products with many development teams vs. small products with few development teams).

In conclusion, we see a high potential of lean practices improving software engineering. However, there is very little work done in this area so far.

7.6 Conclusion

This paper presents a novel method called Software Process Improvement through Lean Measurement (SPI-LEAM). The goals of the method are to 1) enable continuous software process improvement leading to a lean software process; and 2) avoid problems related to resistance of change by improving in a continuous manner. The method combines the quality improvement paradigm with lean measurements.

The method is based on measuring inventories representing different dimensions of software development (normal development work, extra work, and software quality). It was exemplified how the method is used. Feedback from industry and the discussion of the method leads to the following conclusions:

- SPI-LEAM is flexible as it allows companies to choose inventories and analysis methods fitting their needs in the best way. Thus, the method should be applicable in many different contexts.

- The practitioners who reflected on the method agreed on how we approached the problem. They, for example, observed that the inventories should be below the capacity level. Furthermore, they agreed on the need to conduct a combined analysis of inventories to have a complete view of the current situation. That is, the risk of optimizing measures is drastically reduced.

In future work the impact of the method on software process improvement activities is needed. The method focused on the overall process life-cycle. However, the ideas could be useful for a team working on a specific development task, such as the visualization of throughput for a single team, and a measure of cycle-time. This allows each team to determine its own capability level. Furthermore, the analysis of related work showed that generally more research on lean tools is needed.

7.7 References

[1] David Anderson. *Agile management for software engineering: applying the theory of constraints for business results*. Prentice Hall, 2003.

[2] Victor R. Basili. Quantitative evaluation of software methodology. Technical report, University of Maryland TR-1519, 1985.

[3] Victor R. Basili. The experience factory and its relationship to other quality approaches. *Advances in Computers*, 41:65–82, 1995.

[4] Victor R. Basili and Scott Green. Software process evolution at the sel. *IEEE Software*, 11(4):58–66, 1994.

[5] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S. Trivedi. *Queuing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley, Hoboken, N.J., 2. ed. edition, 2006.

[6] Christos G. Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Kluwer Academic, Boston, 1999.

[7] Çigdem Gencel and Onur Demirörs. Functional size measurement revisited. *ACM Transactions on Software Engineering and Methodology*, 17(3), 2008.

[8] CMMI-Product-Team. Capability maturity model integration for development, version 1.2. Technical report, CMU/SEI-2006-TR-008, 2006.

[9] Dan Cumbo, Earl Kline, and Matthew S. Bumgardner. Benchmarking performance measurement and lean manufacturing in the rough mill. *Forest Products Journal*, 56(6):25 – 30, 2006.

[10] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006.

[11] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. A model for software rework reduction through a combination of anomaly metrics. *Journal of Systems and Software*, 81(11):1968–1982, 2008.

[12] Paolo Donzelli and Giuseppe Iazeolla. Hybrid simulation modelling of the software process. *Journal of Systems and Software*, 59(3):227–235, 2001.

[13] John P. Van Gigch. *System design modeling and metamodeling*. Plenum Press, New York, 1991.

[14] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A model for technology transfer in practice. *IEEE Software*, 23(6):88–95, 2006.

[15] Tony Gorschek and Claes Wohlin. Requirements abstraction model. *Requir. Eng.*, 11(1):79–101, 2006.

[16] Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, and Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3):323–332, 2001.

[17] Capers Jones. *Applied software measurement: assuring productivity and quality*. McGraw-Hill, New York, 1997.

[18] Christer Karlsson and Par Ahlströhm. The difficult path to lean product development. *Journal of Product Innovation Management*, 13(4):283–295, 2009.

[19] Mikael Lindvall and Kristian Sandahl. Traceability aspects of impact analysis in object-oriented systems. *Journal of Software Maintenance*, 10(1):37–57, 1998.

[20] Andrea De Lucia, Eugenio Pompella, and Silvio Stefanucci. Assessing effort estimation models for corrective maintenance through empirical studies. *Information & Software Technology*, 47(1):3–15, 2005.

[21] Brian Maskell and Bruce Baggaley. *Practical lean accounting: a proven system for measuring and managing the lean enterprise*. Productivity Press, 2004.

[22] Peter Middleton. Lean software development: two case studies. *Software Quality Journal*, 9(4):241–252, 2001.

[23] Peter Middleton, Amy Flaxel, and Ammon Cookson. Lean software management case study: Timberline inc. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, pages 1–9, 2005.

[24] James M. Morgan and Jeffrey K Liker. *The Toyota product development system: integrating people, process, and technology*. Productivity Press, New York, 2006.

[25] Thane Morgan. Lean manufacturing techniques applied to software development. Master's thesis, Master Thesis at Massachusetts Institute of Technology, April 1998.

[26] Bohdan W. Oppenheim. Lean product development flow. *Systems Engineering*, 7(4):352–376, 2004.

[27] Emma Parnell-Klabo. Introducing lean principles with agile practices at a fortune 500 company. In *Proceedings of the AGILE Conference (AGILE 2006)*, pages 232–242, 2006.

[28] Richard T. Pascale. *Managing on the edge: how the smartest companies use conflict to stay ahead*. Simon and Schuster, New York, 1990.

[29] G.I.U.S. Perera and M.S.D. Fernando. Enhanced agile software development hybrid paradigm with lean practice. In *Proceedings of the International Conference on Industrial and Information Systems (ICIIS 2007)*, pages 239–244, 2007.

[30] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development - state of the art vs. industrial case study. In *Proceedings of the 10th International Conference on Product Focused Software Development and Process Improvement (PROFES 2009)*, pages 386-400, 2009.

[31] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

[32] Donald G Reinertsen. *Managing the design factory: a product developers toolkit*. Free, New York, 1997.

[33] Shigeo Shingo. *Study of Toyota production system from the industrial engineering viewpoint*. Japanese Management Association, 1981.

[34] James P. Womack and Daniel T. Jones. *Lean thinking: banish waste and create wealth in your corporation*. Free Press Business, London, 2003.

[35] Xiaochun Zhu, Bo Zhou, Li Hou, Junbo Chen, and Lu Chen. An experience-based approach for test execution effort estimation. In *Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS 2008)*, pages 1193 – 1198, 2008.

Chapter 8

Measuring the Flow of Lean Software Development

Kai Petersen and Claes Wohlin
Software: Practice and Experience, in print

8.1 Introduction

Agile software development aims at being highly focused and responsive to the needs of the customer [16, 3]. To achieve this practices like on-site customer and frequent releases to customers can be found in all agile practices. Agile practices can be further enhanced by adopting practices from lean manufacturing. Lean manufacturing focuses on (1) the removal of waste in the manufacturing process; and (2) analyzing the flow of material through the manufacturing process (cf. [4, 36, 27]). Both aid the responsiveness to customer needs that agile seeks to achieve. Firstly, removing waste (everything that does not contribute to customer value) frees resources that can be focused an value-adding activities. Secondly, analyzing the flow of development aids in evaluating progress and identifying bottlenecks.

For the removal of waste and the improvement of the flow it is important to understand the current status in terms of waste and flow, which is supported by visualizing and measuring it. Improving the flow means shorter lead-times and thus timely deliver of value to the customer. In manufacturing cumulative flow diagrams have been pro-

posed to visualize the flow of material through the process [29]. Such a flow diagram shows the cumulative number of material in each phase of the manufacturing process. To the best of our knowledge, only one source has proposed to analyze the flow of software development through flow-diagrams by counting customer-valued functions [1]. However, the usefulness of cumulative flow diagrams has not yet been empirically evaluated.

The novel contributions of this paper are: (1) The definition of measures to assess the flow of lean software development with the goals of increasing throughput and creating the transparency of the current status of product development; (2) The evaluation of the visualization combined with the proposed measures in an industrial context.

The case studied was Ericsson AB in Sweden. The units of analysis were nine sub-systems developed at the case company. The structure of the case study design was strongly inspired by the guidelines provided in Yin [37] and Runeson and Höst [31]. The data collection started recently and therefore cannot be used to see long-term trends in the results of the measurements. Though, the case study was already able to illustrate how the measures could influence decision making, and how the measures could be used to drive the improvement of the development flow.

The following measures were defined:

- A measure to detect bottlenecks.
- A measure to detect how continuous the requirements flow through the development process.
- A cost-model separating investment, work done, and waste.

The evaluation of the model showed that practitioners found the model easy to use. They agreed on its usefulness in influencing their management decisions (e.g. when prioritizing requirements or assigning people to tasks). Furthermore, different managers integrated the measures quickly in their work-practice (e.g. using the measurement results in their status meetings). However, the quantification of the software process improvements that can be achieved with the model can only be evaluated when collecting the data over a longer period of time.

The remainder of the paper is structured as follows: Section 8.2 presents related work. Thereafter, Section 8.3 provides a brief overview of cumulative flow diagrams for visualizing the flow as well as the measures identifying bottlenecks, continuity of the requirements flow, and cost distribution. Section 8.4 illustrates the research method used, including a description of the case, a description of how the measures were evaluated, and an analysis of validity threats. The results are presented in Section 8.5, including the application of the measures on the industrial data, and the evaluation

of usefulness of the measures. The practical and research implications are discussed in Section 8.6. Section 8.7 concludes the paper.

8.2 Related Work

The related work covers three parts. First, studies evaluating lean software development empirically are presented. Secondly, measures are presented which are used in manufacturing to support and improve lean production. These are used to compare the measures defined within this case study with measures used in manufacturing, discussing the difference due to the software engineering context (see Section 8.6.3). The third part presents literature proposing lean measures in a software engineering context.

8.2.1 Lean in Software Engineering

The software engineering community became largely aware of lean principles and practices in software engineering through Poppendieck and Poppendieck [27] who illustrated how many of the lean principles and practices can be used in a software engineering context. Lean software development shares principles with agile regarding people management and leadership, the focus on quality and technical excellence, and the focus on frequent and fast delivery of value to the customer. What distinguishes lean from agile is the focus on the end to end perspective of the whole value flow through development, i.e. from very early concepts and ideas to delivery of software features. To support the end to end focus lean proposed a number of tools to analyze and improve the value flow, such as value stream mapping [21], inventory management [1], and pull systems such as Kanban [12]. Value stream mapping visualizes the development life-cycle showing processing times and waiting times. Inventory management aims at limiting the work in process as partially done work does not provide value. Methods to support inventory management are theory of constraints [1] and queuing theory [14]. Push systems allocate time for requirements far ahead and often overload the development process with work. In contrast pull systems allow software teams to take on new tasks whenever they have free capacity. If possible, high priority tasks should be taken first.

Middleton [18] conducted two industrial case studies on lean implementation in software engineering, and the research method used was action research. The company allocated resources of developers working in two different teams, one with experienced developers (case A) and one with less experienced developers (case B). The practitioners responded that initially the work was frustrating as errors became visible almost immediately and were to be fixed right away. In the long run though the number of

errors dropped dramatically. However, after using the lean method the teams were not able to sustain it due to organizational hierarchy, traditional promotion patterns, and the fear of forcing errors into the open.

Another case study by Middleton et al. [19] studied a company practicing lean in their daily work for two years. They found that the company had many steps in the process not adding value. A survey among people in the company showed that the majority supported lean ideas and thought they can be applied to software engineering. Only a minority (10 %) was not convinced of the benefits of lean software development. Statistics collected at the company showed a 25 % gain in productivity, schedule slippage was reduced to 4 weeks from previously months or years, and time for defect fixing was reduced by 65 % - 80 %. The customer response on the product released using lean development was overwhelmingly positive.

Perera and Fernando [24] compared an agile process with a hybrid process of agile and lean in an experiment involving ten student projects. One half of the projects was used as a control group applying agile processes. A detailed description of how the processes differed and which practices were actually used was not provided. The outcome was that the hybrid approach produced more lines of code and thus was more productive. Regarding quality, early in development more defects were discovered with the hybrid process, but the opposite trend was found in later phases, which confirms the findings in [18].

Parnell-Klabo [22] followed the introduction of lean and documented lessons learned from the introduction. The major obstacles were to obtain open office space to locate teams together, gain executive support, and training and informing people to reduce resistance of change. After successfully changing with the help of training workshops and the use of pilot projects positive results were obtained. The lead-time for delivery was decreased by 40% - 50%. Besides having training workshops and pilots and sitting together in open office-landscapes, having good measures to quantify the benefits of improvements are key.

Overall, the results show that lean principles may be beneficial in a software engineering context. Thus, further evaluation of lean principles is needed to understand how they affect the performance of the software process. Though, the studies did not provide details on which lean principles and tools were used, and how they were implemented. None of the studies focused on evaluating specific methods or principles of the lean tool-box. However, in order to understand how specific principles and methods from lean manufacturing are beneficial in software engineering, they have to be tailored and evaluated. This case study makes a contribution by implementing cumulative flow diagrams in industry, defining measures to analyze the development flow, and evaluating the cumulative flow diagrams and defined measures.

8.2.2 Lean Performance Measures in Manufacturing

A number of lean process measures have been proposed for manufacturing. Maskell and Baggaley [17] summarize performance measures for lean manufacturing. As this paper is related to measuring the flow of development we focus the related work on measurement of throughput:

- *Day-by-the-Hour (DbtH)*: Manufacturing should deliver at the rate the customers demand products. This rate is referred to as takt-time. The measure is calculated as

$$DbtH = \frac{\#quantity}{\#hours} \tag{8.1}$$

  where the quantity is the number of items produced in a day, and the hours the number of hours worked to produce the units. The rate should be equal to the takt-rate of demand.

- *Capacity utilization (CU)*: The work in process (WIP) is compared to the capacity (C) of the process. CU is calculated as:

$$CU = \frac{WIP}{C} \tag{8.2}$$

  If $CU > 1$ then the work-load is too high, and if $CU < 1$ then the work-load is too low. A value of 1 is ideal.

- *On-time-delivery (OTD)*: Delivery precision is determined by looking at the number of late deliveries in relation to the total deliveries ordered:

$$OTD = \frac{\#late\ deliveries}{\#deliveries\ ordered} \tag{8.3}$$

8.2.3 Lean Performance Measures in Software Engineering

Anderson [1] presents the measures cost efficiency and value efficiency, as well as descriptive statistics to analyze the flow in software development. From a financial (accounting) perspective he emphasizes that a cost perspective is not sufficient as cost accounting assumes that value is always created by investment. However, this ignores that cost could be waste and does not contribute to the value creation for the customer. In contrast to this throughput accounting is more sufficient to measure performance as it explicitly considers value creation.

In cost accounting one calculates the cost efficiency (CE) as the number of units delivered (LOC) divided by the input person hours (PH). Though, this assumes that there

is a linear relationship between input and output, meaning that the input is invariable [1]. However, developers are not machines as they are knowledge workers. Therefore, this equation is considered insufficient for software production.

$$CE = \frac{\Delta LOC}{PH} \tag{8.4}$$

Therefore, the value created over time is more interesting from a lean perspective, referred to as value efficiency (VE). It is calculated as the difference of the value of output V_{output} and value of input V_{input} within the time-window Δt (see Equation 8.5) [1]. The input V_{input} represents the investment to be made to obtain the unit of input to be transformed in the development process, the value V_{output} represents the value of the transformed input, i.e. final product. Furthermore, V_{input} considers investment in tools and equipment for development.

$$VE = \frac{V_{output} - V_{input}}{\Delta t} \tag{8.5}$$

In addition to throughput accounting Anderson [1] presents descriptive statistics to evaluate lean performance. Plotting the cumulative number of items in inventory in different phases helps determining whether there is a continuous flow of the inventory. How to implement and use this technique as a way to determine process performance in incremental development is shown in the next section.

8.3 Visualization and Measures

This section presents our solution for measuring the flow of lean software development based on cumulative flow diagrams. First, cumulative flow diagrams are introduced and thereafter the measures quantifying the diagrams are presented.

8.3.1 Visualization with Cumulative Flow Diagrams

Cumulative flow diagrams show how many units of production travel through the manufacturing process in a specific time window. In software engineering, it is of interest to know how many requirements are processed in a specific time window for a specific phase of development. A sketch of this is shown in Figure 8.1 with terms from the studied organization.

The x-axis shows the time-line and the y-axis shows the cumulative number of requirements having completed different phases of development. For example, the line on the top represents the total number of requirements in development. The line below

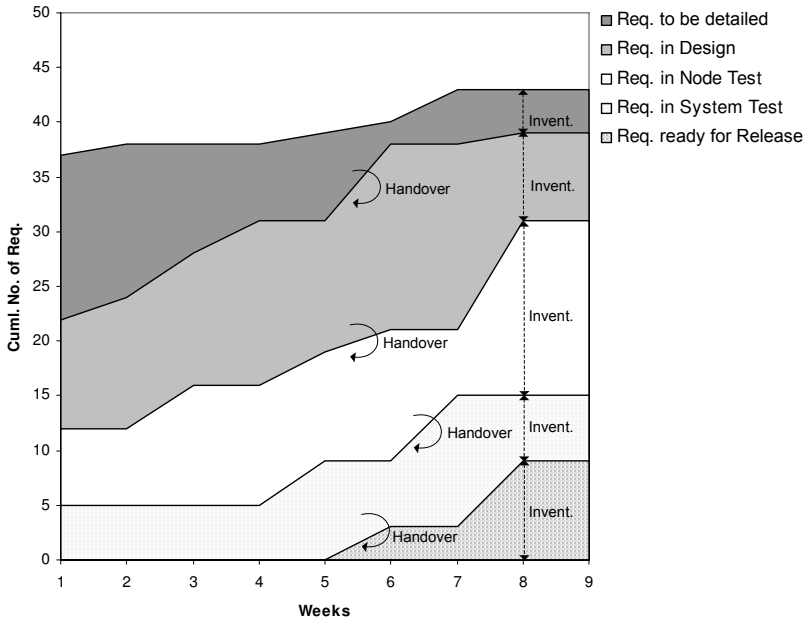


Figure 8.1: Cumulative Flow Diagram for Software Engineering

that represents the total number of requirements for which detailed specification is finished and that were handed over to the implementation phase. The area in between those two lines is the number of incoming requirements to be detailed.

Looking at the number of requirements in different phases over time one can also say that the lines represent the hand-overs from one phase to the other. For example, in week six a number of incoming requirements is handed over to implementation, increasing the number of requirements in implementation. If the implementation of the requirement is finished, it is handed over to the node test (in this case in week 7). In the end of the process the requirements are ready for release.

Inventory is defined as the number of requirements in a phase at a specific point in time. Consequently, the difference of the number of requirements (R) in two phases (j and $j+1$) represents the current inventory level at a specific point in time (t), as shown by the vertical arrows in week 8 in Figure 8.1. That is, the inventory of phase j (I_j) in a specific point in time t is calculated as:

$$I_{j,t} = R_{j,t} - R_{j+1,t} \tag{8.6}$$

8.3.2 Measures in Flow Diagrams

The flow diagrams were analyzed with the aim of arriving at useful performance measures to increase throughput. Before presenting the actual measurement solution we show how we constructed the measures with the aid of the GQM approach [2]. The GQM approach follows a top-down strategy. First, goals were defined which should be achieved. Thereafter, questions were formulated that have to be answered to achieve the goals. In the last step, the measures were identified that need to be collected in order to answer the question. The GQM was executed as follows:

1. The company drove the analysis by identifying the goals to improve throughput in order to reduce lead-time and improve the responsiveness to customer needs.

2. Based on the goals the two authors of the paper individually derived questions and measures considering the goal and having the data of the cumulative flow graphs available. Thereafter, the authors discussed their measures and agreed on the questions and measurements to present to the case company. Both authors identified similar measures.

3. The measures as well as the results of their application were presented to an analysis team responsible for implementing lean measurement at the company. In the meeting with the analysis team the measures were discussed and feedback was given on the measures in an open discussion (see Section 8.4.5). The reception among the practitioners regarding the cumulative flow diagrams was also positive.

The goals identified in the first step were:

G1: increase throughput in order to reduce lead-times and improve responsiveness to customers' needs. Throughput is important due to dynamic markets and rapidly changing customer requirements. This is specifically true in the case of the company we studied. In consequence, started development work becomes obsolete if it is not quickly delivered to the customer.

G2: show the current progress/status of software product development. Showing the current status and progress of development in terms of flow allows management to take corrective actions for improving the flow.

Related to these goals the following questions were used to identify the measures:

Q1: Which phase in the development flow is a bottleneck? A bottleneck is a single constraint in the development process. Resolving this constraint can significantly increase the overall throughput. Bottleneck detection allows to continuously improve throughput by applying the following steps: (1) identify the constraint, (2) identify the cause of the constraint, (3) remove the constraint, (4) go to step (1) [1].

Q2: How even is the workload distributed over time in specific phases? Workload throughout the development life-cycle of the software development process should be continuous. That means, one should avoid situations where requirements are handed over between phases in large batches (see, for example, the hand-over in Figure 8.1 where a large batch is handed over in week seven from implementation to node test). Large batches should be avoided for two main reasons. First, when having large batches there has to be a time of little activity before the batch is handed over. That means, defects introduced during that time are not detected immediately as they have to wait for the hand-over to the next phase for detection. Defects that are discovered late after their introduction are expensive to fix because investigating them is harder [27, 1]. Secondly, one would like to avoid situations where phases (e.g. requirements, coding, or testing) are overloaded with work at one time, and under-loaded at another time. As discussed in [19, 27, 25] early fault detection and a continuous work-load are an integral part of lean as well as agile software development to assure the reduction of waste, and quick responsiveness to customer needs.

Q3: Where can we save cost and take work-load off the development process? This question connects directly to waste, showing in which part of the development life-cycle waste has been produced. The focus of improvement activities should be on the removal of the most significant type of waste.

We would like to point out that RQ1 and RQ2 are not the same thing from a theoretical perspective. It is possible to have a similar throughput of requirements in two phases, but still one phase can deliver continuously while the other delivers in batches. This is further discussed in Section 8.6.

Measures to Quantify Bottlenecks

The derivation of metrics is driven by the questions formulated before. For each question (and by looking at the diagrams) we identified the measures.

Q1: Which phase in the development flow is a bottleneck? A bottleneck would exist if requirements come into one phase (phase j) in a higher rate than they can be handed over to the next phase (phase $j+1$). In Figure 8.1 an example of a bottleneck can be found. That is, the rate in which requirements are handed over from implementation to node test seems to be higher than from node test to system test. This indicates that the node test phase is a bottleneck. To quantify the bottleneck we propose linear regression

to measure the rate of requirements flow for each phase. Linear regression models with two variables are used to predict values for two correlated variables. In our case the variables are *Weeks* and *Cumulative Number of Requirements*. The linear function represents the best fit to the observed data set. To determine the linear function the least square method is commonly used [20]. Furthermore, when the linear regression model is created, it can be observed that there is a difference between the actual observations and the regression line, referred to as the estimation error ε. This leads to the following formula:

$$y = f(x) = \beta_0 + \beta_1 * x + \varepsilon \tag{8.7}$$

When conducting the actual prediction of the parameters, β_0 and β_1 are estimated as those represent the linear regression function. For the analysis of the bottlenecks the predicted variable β_1 is important, representing the slope of the linear functions. The measure for the bottleneck is thus defined as follows: If the slope of phase j (slope is referred to as β_j) is higher than the slope of the subsequent phases (β_{j+p}) then phase j is a bottleneck in the process. Though, it is important to note that the cause of the bottleneck is not necessarily to be found in phase j. To show the results of the measurement to management we propose to draw bar-plots of the slope which are well suited to illustrate the severity of the difference between phases.

Example: Let's assume a situation where the current inventory level of requirement in the specification phase S is 10 ($I_S = 10$). The regression of the incoming requirements of the last four weeks was $\beta_{inc,S} = 3$, and for the outgoing requirements $\beta_{out,S} = 1$. This leads to the following functions over time ($t = weeks$) for incoming and outgoing requirements: $f_{inc,S}(t) = 3t + 10$ and $f_{out,S}(t) = 1t + 10$. If in the following four weeks the rates do not change then the inventory level (and with that work in process) will almost double, as $f_{inc,S}(4) = 22$ and $f_{out,S}(4) = 14$. This leads to a new inventory level of $I_S = 18$. Depending on the available capacity this might lead to an overload situation indicating that actions have to be taken to avoid this rate of increasing inventory.

Measures to Quantify Workload Distribution Over Time

Q2: How even is the workload distributed over time in specific phases? Examples for an uneven work-flow can also be found in Figure 8.1. Continuous flow means that the number of requirements handed over at different points in time varies. For example, high numbers of requirements were handed over from incoming requirements to implementation in weeks two to four and week five, while there is few hand-overs in the remaining weeks. In comparison the variance of the flow of incoming requirements

is lower. In order to quantify how continuous the flow of development is we propose to use the estimation error ε. The estimation error represents the variance around the prediction line of the linear regression, i.e. the difference between the observed and predicted values. The estimation error ε_i is calculated as follows:

$$\varepsilon_i = y_i - \hat{y}_i \tag{8.8}$$

For the analysis the mean estimation error is of interest. In the case of our measurements i in the equation would be the week number. The estimation error should also be plotted as bar-plots when shown to management as this illustrates the severity of differences well.

Example: Let's assume that the variance in hand-overs from the specification phase to the development teams was very high in the last few month based on the average of estimation errors ε. Hence, hand-overs come in batches (e.g. in one week 15 requirements are handed over, while in the previous weeks only 3 requirements were handed over). High variances make the development harder to predict. In a perfectly even flow all observed data points would be on the prediction line, which would only be the case in a completely predictable environment.

Measures of Cost Related to Waste

Q3: Where can we save cost and take work-load off the development process? In order to save costs the costs need to be broken down into different types. We propose to break them down into investment (I), work done (WD), and waste (W) at a specific point in time (i) and for a specific phase (j), which leads to the following equation:

$$C_{i,j} = I_{i,j} + WD_{i,j} + W_{i,j} \tag{8.9}$$

The components of the cost model are described as follows:

- *Investment (I):* Investment is ongoing work that will be delivered to the next phase in the future (i.e. considered for upcoming increments to be released to the market). As long as it is potentially delivered it will be treated as investment in a specific phase j.
- *Work done (WD):* Work done is completed work in a phase, i.e. what has been handed over from phase j to phase $j+1$.
- *Waste (W):* Waste is requirements that are discarded in phase j. That means work has been done on them, but they are not further used in the future; i.e. they will never make it into a release to customers.

An overview of a cost table for one specific point in time is shown in Table 8.1. For each phase j at time i and type of cost (I, WD, and C) the number of requirements is shown. At the bottom of the table the sums of the types of cost is calculated across phases ($j = 1, ...n$).

Table 8.1: Costs

Phase (j)	Investment (I)	Work Done (WD)	Waste (W)	Cost (C)
$j = 1$	$r_{1,I}$	$r_{1,WD}$	$r_{1,W}$	$r_{1,C}$
$j = 2$	$r_{2,I}$	$r_{2,WD}$	$r_{2,W}$	$r_{2,C}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$j = n$	$r_{n,I}$	$r_{n,WD}$	$r_{n,W}$	$r_{n,C}$
all	$\sum_{j=1}^{n} r_{j,I}$	$\sum_{j=1}^{n} r_{j,WD}$	$\sum_{j=1}^{n} r_{jW}$	$\sum_{j=1}^{n} r_{j,C}$

When analyzing a time-frame (e.g. month) the average waste for the time-frame should be calculated. For example, if the points in time $(i, \ldots, m)$ are weeks in a given interval ($i = 1$ being the first week of the interval, and m being the last week) we calculate the average across phases as:

$$C_{avg,all} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,I}}{m} + \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,WD}}{m} + \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} r_{i,j,W}}{m} \tag{8.10}$$

For an individual phase j we calculate:

$$C_{avg,j} = \frac{\sum_{i=1}^{m} r_{i,j,I}}{m} + \frac{\sum_{i=1}^{m} r_{i,j,WD}}{m} + \frac{\sum_{i=1}^{m} r_{i,j,W}}{m} \tag{8.11}$$

This summary allows to observe the average distribution of I, WD, and W within the selected time-frame for the analysis. In order to present the data to management, we propose to also calculate the percentages to show the distribution between types of costs. Besides being a detector for waste, this information can be used as an indicator for progress. If the investment is always much higher than the work-done in consecutive phases that means that the investment is not transferred into a work-product for the investigated phase(s).

Example: Let's assume a cost distribution for the implementation phase of $C_{imp} = 9 + 100 + 200$. The distribution shows that few new requirements are available as input for implementation phase ($I = 9$), and that implementation has completed most of the work ($WD = 100$). Consequently there is a risk that the implementation phase does not

have enough input to work with in the future. Furthermore, the cost distribution shows that the majority of the requirements was discarded ($W = 200$). For the requirements and implementation phase this means that work done (requirements specification, review, coding and unit testing) on the discarded requirements does not end up in the end product. Too much of the cost is put on requirements that are never delivered and it shows that actions have to be taken in the future to minimize these types of costs.

8.4 Research Method

Candidate research methods for our research questions were design science [13], action research [32], and case study [9, 37]. Design science proposes that artifacts (in this case measures) should be created and evaluated. One way to evaluate the artifacts is case study. Hence, design science and case study complement each other. An alternative to case study is action research, also suitable to answer our research questions. However, this would require the researcher to work with the teams and managers using our proposed solution continuously which was not feasible. In consequence, the research method used is an embedded case study allowing to observe and understand the usefulness of the proposed solution. Embedded means that within the case (Ericsson) different embedded units (the process flow for different sub-systems being developed) were analyzed. The design of the case study is strongly inspired by the guidelines provided in Yin [37] and Runeson and Höst [31]. We used Yin's guidelines in identifying the case and the context, as well as the units of analysis. Yin provides an overview of potential validity threats relevant for this case study and stresses the importance of triangulation, i.e. to consult different data sources, such as documentation, quantitative data and qualitative data from workshops, interviews, or observations.

8.4.1 Research Context

Ericsson AB is a leading and global company offering solutions in the area of telecommunication and multimedia. Such solutions include products for telecommunication operators, multimedia solutions and network solutions. The company is ISO 9001:2000 certified. The market in which the company operates can be characterized as highly dynamic with high innovation in products and solutions. The development model is market-driven, meaning that the requirements are collected from a large base of potential end-customers without knowing exactly who the customer will be. Furthermore, the market demands highly customized solutions, specifically due to differences in services between countries. The following agile practices are used: continuous integration, internal and external releases, time-boxing with sprints, face-to-face interaction

(stand-up meetings, co-located teams), requirements prioritization with metaphors and detailed requirements (digital product backlog), as well as refactoring and system improvements. Version control is handled with ClearCase [33] and TTCN3 [34] is used for test automation. Documentation and faults are tracked in company proprietary tools.

8.4.2 Case Description

The case being studied was Ericsson in Sweden and India. On a high level all systems were developed following the same incremental process model illustrated in Figure 8.2. The numbers in the figure are mapped to the following practices used in the process.

- *Prioritized requirements stack (1) and anatomy plan (2):* The company continuously collects requirements from the market and prioritizes them based on their importance (value to the customer) and requirements dependencies (anatomy plan). Requirements highest in the priority list are selected and packaged to be implemented by the development teams. Another criterion for packaging the requirements is that they fit well together. The anatomy plan also results in a number of baselines (called last system versions, LSV) and determines which requirement packages should be included in different baselines.

- *Small projects time line (3):* The requirements packages are handed over to the development projects implementing and unit testing the increments of the product. The projects last approximately three months and the order in which they are executed was determined in the previous two steps.

- *Last system version (4):* As soon as the increment is integrated into the sub-system a new baseline is created (LSV). Only one baseline exists at one point in time. The last version of the system is tested in predefined testing cycles and it is defined which projects should be finished in which cycle. When the LSV phase has verified that the sub-system works and passed the LSV test with the overall system the sub-system is ready for release.

- *Potential release (5):* Not every potential release has to be shipped to the customer. Though, the release should have sufficient quality to be possible to release to customers.

Overall the process can be seen as a continuously running factory that collects, implements, tests, and releases requirements as parts of increments. When a release is delivered the factory continuous to work on the last system version by adding new

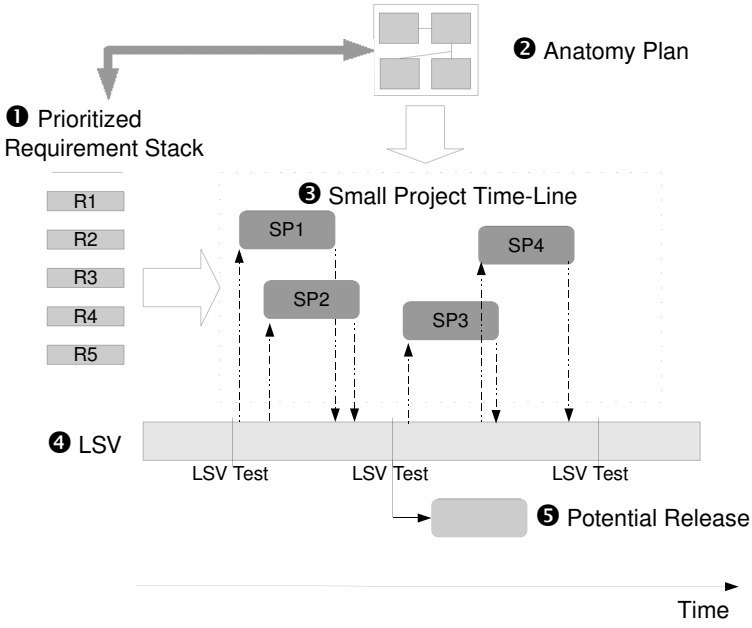


Figure 8.2: Incremental Process Model

increments to it. The sub-systems used for illustration in the paper have been around for more than five years and have been parts of several major releases to the market.

8.4.3 Units of Analysis

In total the flow of nine sub-systems developed at the case company were analyzed with the flow diagrams, each sub-system representing a unit of analysis. Two of the sub-systems are presented in this paper representing different patterns. The difference in flows helps (1) to identify patterns that are worthwhile to capture in the measures; and (2) to derive more general measures that can be applied to a variety of flows. In addition to the two individual sub-systems we conducted a combined analysis of all nine sub-systems studied at the case company. The sub-systems have been part of the system for more than five years and had six releases after 2005. The sub-systems vary in complexity and the number of people involved in their development.

1. *Unit A*: The sub-system was developed in Java and C++. The size of the system

was approximately 400,000 LOC, not counting third party libraries.

2. *Unit B*: The sub-system was developed in Java and C++, the total number of LOC without third party libraries was 300,000.

The nine sub-systems together had more than 5,000,000 LOC. The development sites at which the sub-systems were developed count more than 500 employees directly involved in development (including requirements engineering, design, and development) as well as administration and configuration management.

8.4.4 Research Questions

We present the research questions to be answered in this study, as well as a motivation why each question is of relevance to the software engineering community.

- *Research question 1 (RQ1): Which measures aid in (1) increasing throughput to reduce lead-times, and (2) as a means for tracking the progress of development?* The aim is to arrive at measures that are generally applicable and suitable to increase throughput and track the progress of development. As argued before, throughput and lead-times are highly important as customer needs change frequently, and hence have to be addressed in a timely manner. Thus, being able to respond quickly to customer needs is a competitive advantage. In addition knowing the progress and current status of complex software product development should help to take corrective actions when needed, and identify the need for improvements.

- *Research question 2 (RQ2): How useful are the visualization and the derived measures from an industrial perspective?* There is a need to evaluate the visualization (cumulative flow diagrams) and the measures in industry to provide evidence for their usefulness. For this purpose the following sub-questions are addressed in the evaluation part of this study:

  - *Research question 2.1: How do the visualization/measures affect decision making?*
  - *Research question 2.2: What improvement actions in the development processes do practitioners identify based on the visualization and measures?*
  - *Research question 2.3: How should the visualization and the measures be improved for practical use?*

8.4.5 Data Collection

Collection of Data for Visualization and Measurements

The current status (i.e. the phase in which the requirements resides) was maintained in a database. Whenever the decision was taken to move a requirement from one phase to the next this was documented by a time-stamp (date). With this information the inventory (number of requirements in a specific phase at any point in time) could be calculated.

After the initial data entry documenting the current status of requirements the main author and the practitioners reviewed the data for accuracy and made updates to the data when needed. From thereon the practitioners updated the data continuously keeping it up-to-date. To assure that the data was updated on a regular basis, we selected practitioners requiring the status information of requirements in their daily work and who were interested in the visualization results (cumulative flow diagrams) and measurements. The following five inventories were measured in this study:

- *Number of incoming requirements:* In this phase the high level requirements from the prioritization activity (see practice 1 in Figure 8.2) have to be detailed to be suitable as input for the design and implementation phase.
- *Number of requirements in design:* Requirements in this phase need to be designed and coded. Unit testing takes place as well. This inventory represents the requirements to be worked on by the development teams, corresponding to practice 3 in Figure 8.2. After the requirements are implemented they are handed over to the LSV test for system testing.
- *Number of requirements in LSV test (node and system):* These inventories measure the number of requirements that have to be tested in the LSV. The LSV test is done in two steps, namely node LSV (testing the isolated sub-system) and system LSV (testing the integration of sub-systems) measured as two separate inventories. When the LSV test has been passed the requirements are handed over to the release project. This inventory corresponds to practice 4 in the process shown in Figure 8.2.
- *Number of requirements available for release:* This inventory represents the number of requirements that are ready to be released to the customer. It is important to mention that the requirements can be potentially released to the customer, but they do not have to (see Practice 5 in Figure 8.2). After the requirements have been released they are no longer in the inventories that represent ongoing work.

Only ongoing work was considered in the analysis. As soon as the requirements were released they were removed from the diagrams. The reason is that otherwise the number of requirements would grow continuously, making the diagrams more or less unreadable and harder to compare over time.

Collection of Qualitative Data on Usefulness of Visualization and Measures

The result of the evaluation answered research question 2, including the two sub-questions to be answered (research question 2.1 and 2.2).

Research question 2.1: In order to answer research question 2.1 a workshop was conducted by an external facilitator (a consulting company represented by three consultants running the workshop), where the researcher acted as a participant and observer in the workshop. In addition to identifying the effect of the measures on the decision making in the company, the workshop was also used to reflect on possible improvements and measures complementary to the ones identified in this study. During the workshop the participants first noted down the roles that are affected by the measures. Those were clustered on a pin-board and the roles were discussed to make sure that everyone had the same understanding of the responsibilities attached to the identified roles. Thereafter, each participant noted down several effects that the measures have on the decision makers. The effects were discussed openly in the workshop. The researcher took notes during the workshop. In addition to that the external facilitator provided protocols of the workshop session.

Research question 2.2 and 2.3: These research questions were answered by participating in regular analysis meetings run by the company once or twice a month. The purpose of these meetings was to reflect on the usefulness of the measures as well as on the actual results obtained when measuring the flow of the company. During the meetings it was discussed (1) how the measurement results can be interpreted and improved, and (2) what improvement actions can be taken based on the measurement results. The researcher took an active part in these discussions and documented the discussions during the meetings.

The roles participating in the workshop and the meetings were the same. The participants of the meeting and workshop all had management roles. An overview of the roles, and the number of participants filling out each roles are shown in Table 8.2. The criteria for selecting the participants of the workshop and meetings were (1) good knowledge of the processes of the company, and (2) coverage of different departments and thus disciplines (i.e. requirements, testing, and development).

Table 8.2: Roles

Role	Description	No. of Persons
Process Improvement Driver	Initiate and Monitor SPI activities	2
Project Manager	Motivate teams, control projects, reporting	3
Program/Portfolio Manager	Prioritize implementation of requirements, request staffing for program development	2
Line Manager	Allocation of staff, planning of competence development	4

8.4.6 Data Analysis

The data analysis was done in three steps. In the first step the yellow notes were clustered on a high level into groups for statements related to the effect of the measures on decision making (RQ2.1), improvement actions identified based on the measures (RQ2.2), and improvements to measurements (RQ2.3). The clustering in the first step was done in the workshop. Within these groups further clusters were created. For example, clusters in the group for RQ2.1 were requirements prioritization, resources and capacity, and short-term as well as long-term improvements in decisions. The clusters were also compared to the notes of the facilitator running the workshop (see data collection approach for research question 2.1 in Section 8.4.5) to make sure that similar clusters were identified. For the meetings (see data collection approach for research question 2.2 and 2.3) the researcher and a colleague documented the outcome of the meetings, which also allowed to compare notes. In the second step the notes taken during the workshop and meetings were linked to the clusters. Based on these links each cluster was narratively described by the researcher. In the third and final step the analysis was reviewed by a colleague at the company who also participated in the workshop and meetings.

8.4.7 Threats to Validity

Threats to validity are important to consider during the design of the study to increase the validity of the findings. Threats to validity have been reported for case studies in Yin [37] and in a software engineering context in Wohlin et al. [35]. Four types are distinguished, namely construct validity, internal validity, external validity, and reliability. Internal validity is concerned with establishing a causal relationship between variables. External validity is concerned with to what degree the findings of a study can be generalized (e.g. across different contexts). Reliability is concerned with the replication of the study (i.e. if the results would be the same when repeating the study).

Internal validity is not relevant for this study as we are not seeking to establish the casual relationship between variables in a statistical manner.

Construct validity: Construct validity is concerned with obtaining the right measures for the concept being studied. There is a risk that the researcher influences the outcome of the study with his presence in industry (reactive bias) [35]. This risk was reduced as the researcher is not perceived as being external as he is also employed by the company. Thus, the participants of the workshops and meetings did not perceive the researcher as an external observer and hence their behavior was not influenced by his presence. Another threat is the risk that the practitioners misinterpret the measurements and visualizations (incorrect data). In order to minimize the threat of incorrect data a tutorial was given on how to read and interpret the visualization and measurements. Furthermore, tutorial slides were provided to all users of our proposed solution. The practitioners were also given the possibility to ask clarification questions which aided in achieving a common understanding of the solution.

External validity/generalizability: The main threat to external validity is that the study has been conducted in a single company. Consequently, the results were obtained in the unique environment in which the case is embedded, namely the context [26, 30, 37]. The threat to external validity was reduced by describing the context carefully (see Section 8.4.1) and by that make the degree of generalizability explicit [25, 26]. Hence, the results of this study are likely to be generalizable to companies working with large-scale development in an incremental and iterative manner, and developing software for a dynamic mass market. In order to employ our solution in other contexts the requirements inventories might differ depending on which other phases of development can be found in another company. In summary, it is always very hard to generalize a case study. However, the findings and experiences gained are expected to provide valuable inputs to others interested in applying a similar approach, although the specific context has always to be taken into account.

Reliability: When collecting qualitative data there is always a risk that the interpretation is affected by the background of the researcher. The threat is reduced because the notes of the researcher were compared to the notes of the external facilitator for the workshop, and with notes from a colleague for the meetings (see Section 8.4.5). Additionally, the analysis was reviewed by a colleague at the company who also participated in the workshop and meetings. Possible misunderstandings are further reduced due to that the researcher has good knowledge of the processes at the company and thus understands the terminology (cf. [5]). The comparison of the notes showed that there were no conflicting statements in the notes.

8.5 Results

The quantitative results (visualization and measurements) are illustrated first. Thereafter, the qualitative evaluation of the usefulness of visualization and our proposed measures is presented.

8.5.1 Application of Visualization and Measures

We applied the measures on data available from the company. The following data is provided for this: (1) the original graphs including the regression lines to visualize the relation between original data and metrics collected; (2) bar-plots of the slope to illustrate the bottlenecks; (3) bar-plots of the estimation error to illustrate how even the flow is; (4) a table summarizing costs as I, WD, and W.

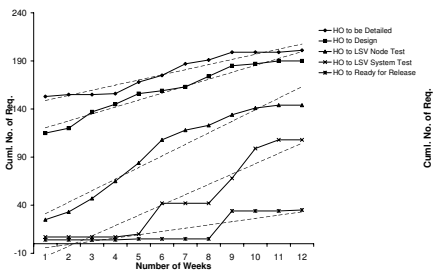
Bottlenecks and Even Flow

Figure 8.3 shows the cumulative flow diagrams for all nine sub-systems combined and the considered units of analysis, including the regression lines. As discussed before the regression lines are a measure of the rate in which requirements are handed over between the phases. The purpose of the figures is to show how the visualization of the original data (cumulative flow diagrams) and the metrics collected are connected. Drawing the regression lines in combination with the original data has advantages.

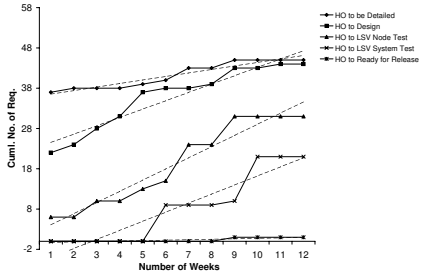
Measurement makes visualization more objective: The sole visualization of the cumulative flow diagram does not always reveal the real trend as it is not easy to recognize which of the phases that has a higher rate of incoming requirement. A high variance around the regression line makes this judgment even harder. Therefore, it is important to calculate the regression. An example is shown in Figure 8.3(b), where the curves for requirements to be detailed (HO to be Detailed) and requirements to be implemented (HO to design) have a fairly similar trend. However, the best fit of the data (regression) makes explicit that the slope of requirements to be detailed is higher than the slope of requirements to be implemented, which (according to our measures) would be an indicator for a bottleneck. Thus, the measurement can be considered more objective in comparison to the sole visualization.

Identifying bottlenecks: All figures show examples of bottlenecks, the main bottleneck (for the designated time-frame) was the node testing phase (i.e. slope HO to LSV Node test $>$ slope HO to LSV System test). The figures also show opposite trends to bottlenecks, i.e. the rate in which requirements come into a phase was lower than the requirements coming out of the phase (e.g. slope HO to Design $<$ slope HO to LSV Node Test 8.3(c)). Such a trend can be used as a signal for recognizing that there is not

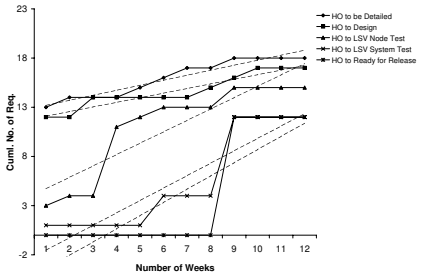
enough investment (or buffer) in this phase to make sure that the forthcoming phase has input to work with in the future. Thus, there is potentially free (and unused) capacity within the process. In order to show the significance of bottlenecks to management, we proposed to draw bar-plots. The bar plots illustrating the slope (value of β) for the different hand-overs and systems are shown on the left side of Figure 8.4. The hight of the bars shows the significance of the bottlenecks. For example, for all systems (Figure 8.4(a)) as well as unit A (Figure 8.4(c)) and B (Figure 8.4(c)) the rate in which requirements are handed over to system test was almost four times higher than the rate the requirements are handed over to the release, which indicates that the work in process before system test will grow further. In consequence an overload situation might occur for system test. In comparison, the bottleneck in the LSV node test was less significant in Figures 8.4(a) and 8.4(c). It is also important to mention that a high slope is desired, which implies high throughput.



(a) All Nine Sub-Systems



(b) Unit A



(c) Unit B

Figure 8.3: Regression Analysis

Evaluating even flow: The highest variances can be found in the HO to LSV system test for all systems, as shown in the graphs on the right side of Figure 8.4. That is, there

was a high deviation between the regression line and the observed data. As discussed before, this indicates that a higher number of requirements was delivered at once (e.g. for all systems quite a high number was delivered between week 8 and 11, while there was much less activity in week 1-7, see Figure 8.3). Figures 8.4(b), 8.4(d) and 8.4(f) show the significance of the variances. It is clear that the highest variances can be found in the implementation and testing phases, while the requirements phase is characterized by lower variances. This is true for the situation in all three figures (8.4(b), 8.4(d) and 8.4(f))

Lower variances can be found in the phases where the requirements are defined (i.e. HO to be detailed, and HO to Design).

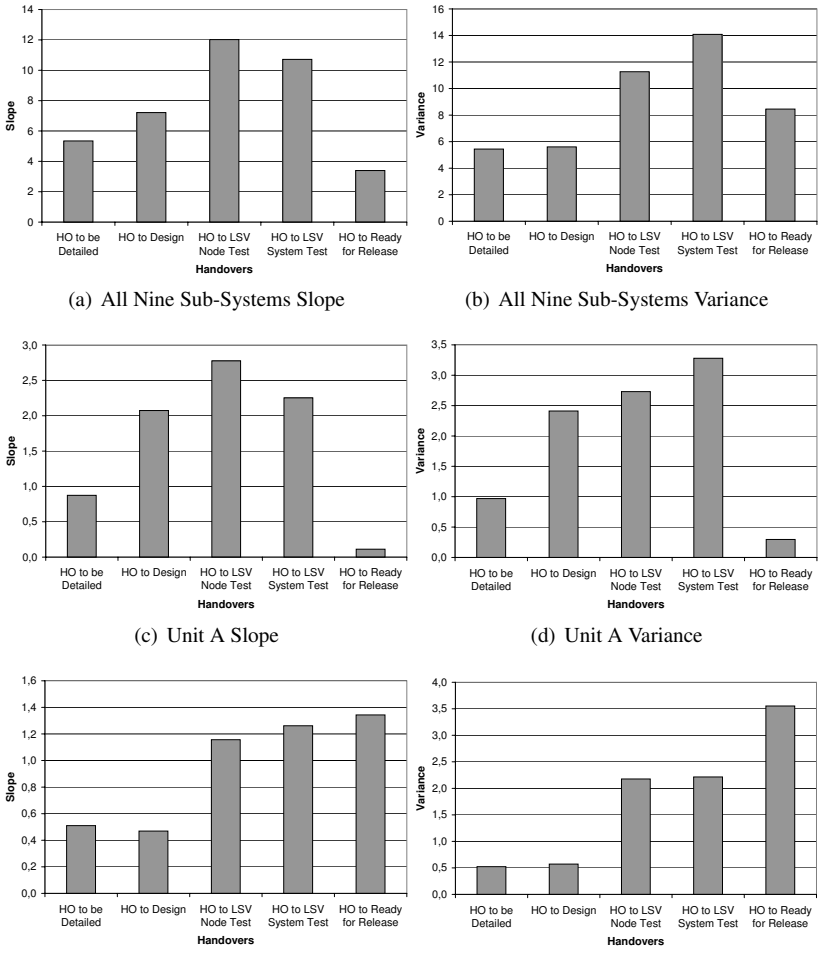
Distribution of Costs

The distribution of costs for all nine sub-systems is shown in Table 8.3. The costs were calculated for each phase ($C_{avg,j}$) and across phases ($C_{avg,all}$). The value of waste (W) was 0 in all phases as no requirements were discarded for the analyzed data sets. The data in the table shows that in the early phases (incoming requirements till requirements to be defined), there was little investment (I) in comparison to work done (WD). That means that investments were transferred into work-done, which is an indicator of progress. However, if there is only little investment in a phase it is important to create new investments (e.g. increasing the number of requirements to be detailed by focusing more on requirements elicitation). Otherwise, the company might end up in a situation where their developers and testers are not utilized due to a lack of requirements (although highly unlikely). From a progress perspective, this analysis looks positive for the early phases. The later phases indicate that the investment was higher than the work done. That means, from a progress perspective most investments in the phase LSV node and LSV System still have to be transferred to work done.

In the long run we expect requirements to be discarded (e.g. due to changes in the needs of customer, or that requirements are hold up in a specific phase). If waste becomes a significant part of the cost distribution, then the reasons for this have to be identified. For example, only requirements with a certain priority should be transferred to a future phase to assure their timely implementation.

8.5.2 Industry Evaluation of Visualization and Measures

Two main aims were pursued with the evaluation. Firstly, we aimed at identifying the relevant roles in the organization that could make use of the measures in their decision making. This first aim helps understanding whether the measures are of use, and



(a) All Nine Sub-Systems Slope

(b) All Nine Sub-Systems Variance

(c) Unit A Slope

(d) Unit A Variance

(e) Unit B Slope

(f) Unit B Variance

Figure 8.4: Slope and Variance

to whom they are most useful (Research Question 2.1). Secondly, we aimed at identifying process improvement actions based on the measures (Research Question 2.2). The second aim shows that the measures trigger practitioners in identifying potential improvement areas/actions, which is an indicator for that the measures serve as an ini-

Table 8.3: Costs All

Phase	I	I (%)	WD	WD (%)	W
$C_{avg,Inc.Req.}$	18.00	10.15	160.00	89.85	0.00
$C_{avg,Design}$	62.92	39.30	97.17	60.70	0.00
$C_{avg,LSVNode}$	51.58	53.09	45.58	46.91	0.00
$C_{avg,LSVSys}$	31.17	68.37	14.42	31.63	0.00
$C_{avg,ReadyRelease}$	14.42	100.00	0.00	0.00	0.00
$C_{avg,all}$	35.63	35,69	63.45	64.04	0.00

tiator for software process improvement activities.

The Effect of Measurements on Decision Making (RQ2.1)

During the workshops with the company representatives, it was discussed how the measures can affect decision making in the company. In the following the different types of decisions supported by the measures are discussed.

Requirements prioritization: The measure of continuous work-flow may show a trend where too many requirements are handed over at once. Thus, the measure helps in assuring that requirements are not handed over in big bulks, but instead in smaller chunks and more continuously. Furthermore, if the measure signals that the development is overloaded, no more requirements should be handed over to avoid an overload situation. As the decisions are related to requirements prioritization the measures help improving from a short-term perspective.

Staff allocation: In case of bottlenecks managers can use the information from the bottleneck analysis to allocate staff when bottlenecks occur. As discussed during the workshop, this has to be done with care, as the reason for a bottleneck might be something else than staffing. However, if staffing is the problem the figures support managers in arguing for the need of staff. Staffing can also be seen as an ad-hoc solution with a short-term perspective.

Transparency for teams and project managers: The measurement system allows seeing which requirements are coming into phases, which are currently worked on, and which have been completed. In consequence, the current status of development for each team is always visible. For instance, the testing team can see all requirements they are supposed to be working on in the upcoming weeks. This helps them in planning their future work as they are aware of future requirements. Furthermore, seeing what has been completed has a motivating effect on the teams.

Software process improvement: Software process improvement needs to look at the data from a long-term perspective (i.e. quarterly is not long enough, instead one should use data for 6 or 12 months). If there are significant bottlenecks or uneven requirements flows the causes for this have to be investigated. Thus, from a process improvement perspective focusing on long-term improvement the data should be used as an indicator of where to look for problem causes.

The result of the workshop was that the practitioners judged the measurements as useful for different roles in the organization, which supports their practical relevance. Furthermore, the measures can be used to take actions to improve in the short and the long term.

Process Improvement Actions Based on Measurements (RQ2.2)

In the analysis team meetings, process improvement actions were identified from a software process improvement perspective baed on the measures. The analysis was done on quarterly data as the data for half a year is not available yet. However, the analysis meetings already show that the information provided by the visualization and measurements aids the practitioners in identifying concrete areas for improvement. The most important improvement areas identified so far are (1) avoid to be too much driven by deadlines and market-windows; and (2) integrate more often to improve quality.

Observation and improvement for (1): The development is done continuously, but with a deadline (market-window) in mind. This is a way of thinking that leads to that work is started late before the deadline (e.g. the high amount of requirements handed over in week 8 to LSV system test in Figure 8.3(c), and the related high value in variance seen in Figure 8.4(f)). In consequence, there were fewer deliveries of requirements from one phase to the other until shortly before the deadline, where a big bulk of requirements was delivered at once. As an improvement we found that people should not focus on deadlines and market-windows too much when planning the flow of development. Instead, they should focus more on the continuous production with optional releases. In order to achieve this, an option is to follow the Kanban approach. In Kanban, the work (e.g. maximum number of requirements with similar complexity) that can be in process at a specific point in time is limited [12]. In consequence, big bulk deliveries of requirements are not possible. Thus, the Kanban approach enforces to continuously work and deliver requirements between phases, reducing the variance in the flow.

Observation and improvement for (2): The practitioners observed that the system was integrated too late, and not often enough. An indication for this observation can be seen in the slope analysis of in Figures 8.4(a) and 8.4(c). Hence, the planning of testing cycles needs to enforce short periods between integration. The main purpose of this is

to allow for regular feedback on the builds helping the company to further improve the quality of the products. In order to technically enable more frequent integration the company is pursuing a higher degree of testing automation.

Improvements to the Measures (RQ3.3)

The visualization and measures need to be further improved in order to increase their usefulness and accuracy. Together with the practitioners providing a critical reflection on the visualization and measures we identified the following improvements:

Treatment of quality requirements: Quality requirements do not flow through the development process in the same fashion as functional requirements. Functional requirements are part of an increment, and are completed with the delivery of the increment. Some quality requirements (e.g. performance) are always relevant for each new release of the product and thus always stay in the inventories. That means, one has to be aware that those requirements are not stuck in the inventory due to a bottleneck. Therefore, we recommend to remove requirements from the analysis that are always valid for the system.

Requirements granularity and value: If requirements vary largely in complexity it is not a valid approach to only count the requirements. One has to take the weight of each requirement into consideration. The weight should be based on the size of the requirements. We propose to estimate the size of the requirements in intervals for small, medium, and large requirements. What small, medium, and large means differs between organizations, meaning that each organization has to determine their own intervals. When counting the requirements, a small requirement is counted once, a medium requirement twice, and a large requirement thrice. Another distinguishing attribute of requirements is their value as suggested by the concept of minimum marketable requirements [7]. High value requirements should flow through the process more quickly and with higher throughput than low value requirements. Consequently, the solution would benefit from also weighting requirements based on their value.

Optimization of measures: Optimization of measures is always a risk when measurements are used to evaluate an organization. For example, it is possible to improve the rate of requirements hand-overs by delivering detailed requirements with lesser quality to implementation. We believe that this will be visible in the measures as design will not be able to work with the requirements and thus a bottleneck in design becomes visible. However, it is still beneficial to complement the evaluation of flow with other quality related measures. Possible candidates for quality measures are fault-slip through [6], or number of faults reported in testing and by the customer.

Visualization of critical requirements: The visualization does not show which requirements are stuck in the development process. In order to requirements into account

that are stuck in the process one could define thresholds for how long a requirement should stay in specific phases. Requirements that are approaching the threshold should be alerted to the person responsible for them. In that way, one could pro-actively avoid that requirements do not flow continuously through the development.

Time frames: The time-frame used at the company for evaluating the flow is each quarter. As discussed with the practitioners, it is also important to have a more long-term perspective when evaluating the data (i.e. half-yearly and yearly). The data in the short-term (quarterly) is more useful for the teams, program/line-managers, and requirements/portfolio managers, who use the data from a short-term perspective to distribute resources or prioritize requirements (see answers to RQ2.1). However, the practitioners responsible for long-term improvements require measurements over a longer period of time to reduce the effect of confounding factors. Examples for confounding factors are Midsummer in Sweden, upcoming deadline of an important release, or critical fault report from customer.

The next section discusses practical implications and research implications of the visualization and measurements presented. Furthermore, the measures are compared to those presented in the related work section, which leads to a discussion of the difference in measuring lean manufacturing and lean software development.

8.6 Discussion

The application and evaluation of the visualization and the defined measures lead to some implications for both research and practice. In addition, we discuss how the measures relate to lean measurements used in manufacturing.

8.6.1 Practical Implications and Improvements to the Measures

The implications describe the relevance of the visualization and measures to industry and what evidence in the results (Section 8.5) supports the relevance.

Perceived usefulness of visualization and measures: The evaluation showed that the visualization and measures are useful from an industrial point of view. The measures were introduced in February 2009 and are mandatory to use in the evaluations of product development of the systems considered in this case study since July 2009 for the studied system developed in Sweden and India. The reason for this is that the measures were able to give a good overview of progress. This kind of transparency is especially important in complex product development where many tasks go on in parallel (see Chapter 3). The rapid application of the solution and the commitment of the

company to use the solution in their daily work is strong evidence of its perceived usefulness. The evaluation of the measures with regard to usefulness was very much based on the perception and feedback by the practitioners. From past experience we found that when transferring a new practice to industry the feedback and commitment from the practitioners is invaluable, the reason being that the practitioners can influence and improve the approach (as has been documented in the improvement suggestions made by the practitioners) [11, 10]. Furthermore, a buy-in by the management is important for a long-term success, which can be achieved through feedback and building trust by incorporating that feedback.

Visualization and measures drive practitioners towards lean and agile practices: The proposed process improvement actions are an indication that the measures drive the organization towards the use of lean and agile practices. The evidence for the claimed driving force of the measures is the identification of lean and agile improvements that the company would like to adopt (see Section 8.5.2). The first improvement proposed is directly related to lean principles (Kanban) [12], stressing the importance to change from a push to a pull mechanism, and by that limiting the work in process [27]. The second improvement stresses the importance of frequent integration and early feedback to improve quality, which is an important principle in all agile development paradigms [3, 25]. Currently actions are in implementation at the company to support the pull approach from the requirements phase to development, and to enable earlier testing (e.g. changes in architecture to allow for continuous upgrades and earlier testing of these upgrades). The actual improvements will not be visible in the visualization and measurements immediately as a large-scale organization with more than 500 people directly and indirectly involved in the development of the investigated systems has been studied. Lean is a new way of thinking for many people in the organization who have to familiarize with the concepts, requiring training and getting a commitment to this new way of working throughout the organization. As pointed out by [23, 18] the introduction of lean software development is a continuous process and the attempt of a big-bang introduction often leads to failure.

Measures help in short-term and long-term decision making: The practitioners perceive the visualization and measurements as beneficial in supporting them in the short and the long term. Evidence is the identification of decisions by the practitioners that were related to long as well as short term decisions. (see Section 8.5.2). Three decisions were related to short term, i.e. requirements prioritization, staff allocation, and project management and planning decisions. Furthermore, undesired effects seen in the measures help in making decisions targeted on long-term process improvements. A prerequisite to take the right long-term decisions is to identify the causes for the undesired effects.

Understand complexity: The large system studied is highly complex with many

different tasks going on in parallel, which makes transparency and coordination challenging [25]. Hence, measurement and visualization focusing on the end to end flow increases the transparency for the practitioners. In particular our solution was able to show the behavior of the development flow of a highly complex system in terms of bottlenecks, continuous flow, and distribution of costs.

Overall, the discussion shows that the results indicate that the measures can be useful in supporting software process improvement decisions.

8.6.2 Research Implications

The research implications provide examples of how other researchers can benefit from the results in this study.

Complement agile with lean tools: The characteristic that distinguishes lean and agile development is the focus on the flow of the overall development life-cycle. Lean provides analysis and improvement tools focusing on the overall development life-cycle while agile focuses on solutions and prescribes sets of practices to achieve agility. Given the results of the case study we believe that the agile research community should focus on making lean analysis tools for the development flow an integral part of agile practices. We believe this to be particularly important in large-scale software development where many teams develop software in parallel, making it harder to achieve transparency of what is going on in a dynamic and agile environment.

Use solution to evaluate process improvements: The measures provided are potentially useful for researchers who would like to evaluate and test improvements increasing efficiency of development. For example, simulation models often focus on lead-time (cf. [8, 28]). Complementary to that the simulations could benefit from making use of the measures identified in this study to provide a solution allowing to analyze throughput with regard to bottlenecks, variance in flow, and distribution of cost.

8.6.3 Comparison with State of the Art

In the related work section, we presented measures that are applied in measuring lean manufacturing. However, in the context of software engineering the measures have drawbacks.

Throughput measures: Throughput measures calculate the output produced per time-unit. Three throughput measures have been identified in the related work, namely day-by-the-hour [17], cost efficiency [1], and value efficiency [1]. In the context of software engineering, this measure would determine the number of requirements completed per hour. However, this measure is very simplistic and does not take into account

the variance in which requirements are completed. Therefore, we use regression in order to calculate the rate and also calculate the variance to evaluate how continuous the flow of development is.

Capacity utilization: In manufacturing it is predictable how many units a machine can produce, and thus the desired value of capacity utilization is clear (i.e. $CU = 1$) [17]. However, software developers are knowledge workers and their behavior is not as predictable. There is a high variance between developers in terms of productivity [15]. Furthermore, when thinking about concepts and creative solutions no output is produced in this time. This makes the measure unsuitable in the software engineering context.

On-time-delivery: On-time delivery [17] is tightly connected to deadlines in software development. However, in the case of incremental development one should not focus too much on a specific deadline, but on being able to continuously deliver a product with the highest priority requirements implemented [27].

The analysis of the related work show that there were no comprehensive measures to capture the flow of development. We addressed the research gap by proposing measures to detect bottlenecks, discontinuous flow, and distribution of costs. The measures are connected to an easy to understand visualization of the development flow, which aids in communicating with management.

8.7 Conclusion

In this study we applied cumulative flow diagrams to visualize the flow of requirements through the software development life-cycle. The main contribution of this study is a set of measures to achieve higher throughput and to track the progress of development from a flow perspective. The measures were evaluated in an industrial case study. In the following we present the research questions and the answers to the questions.

RQ1: Which measures aid in (1) increasing throughput to reduce lead-times, and (2) as a means for tracking the progress of development? Three metrics were identified in the context of this study. The first metric allows to identify bottlenecks by measuring the rate of requirements hand-over between different phases through linear regression. The second metric measures the variance in the hand-overs. If the variance of hand-overs is very high then big batches of requirements are handed over at once, preceded by a time-span of inactivity. The third metric separates requirements into the cost types investment, work done, and waste. The purpose of the measure is to see the distribution of requirements between the different types of cost.

RQ2: How useful are the visualization and the derived measures from an industrial perspective? This research question was evaluated from two angles. First, we evaluated

how the measures can affect decision making. The findings are that: (1) requirements prioritization is supported; (2) the measures aid in allocating staff; (3) the measures provide transparency for teams and project managers of what work is to be done in the future and what has been completed; and (4) software process improvement drivers can use the measures as indicators to identify problems and achieve improvements from a long-term perspective. Secondly, we evaluated what improvement actions practitioners identified based on the measurements. The improvement areas are: (1) an increased focus on continuous development by limiting the allowed number of requirements in inventories; and (2) earlier and more frequent integration and system testing of the software system to increase quality. The solution has been successfully transfered to industry and will be continuously used at the company in the future.

In conclusion the case study showed that the visualization and measures are perceived as valuable from an industrial perspective. It should be emphasized that they are especially valuable when developing large scale products with many teams and tasks going on in parallel, as here transparency is particularly important.

Future work should focus on evaluating our solution in different contexts and how it has to be adjusted to fit these contexts. For example, cumulative flow diagrams and the suggested measures could be applied to analyze software maintenance, or software testing processes. In addition future work should focus on the analysis of what type of improvements support a lean software process, the measurements and visualizations proposed in this paper can be an instrument to evaluate such improvements from a lean software engineering perspective. Little is also known about the long-term effect of implementing agile practices such as Kanban, which is also an interesting area for future research.

From an analysis perspective other lean tools are available, such as value stream maps and theory of constraints/ queuing theory. The usefulness of these tools will allow to learn more about the benefits that could be achieved when using lean practices in the software engineering context. As the approaches aim for similar goals (i.e. the identification of waste) the approaches should be compared empirically to understand which of the approaches is the most beneficial.

8.8 References

[1] David Anderson. *Agile management for software engineering: applying the theory of constraints for business results*. Prentice Hall, 2003.

[2] Victor R. Basili. Quantitative evaluation of software methodology. Technical report, University of Maryland TR-1519, 1985.

[3] Kent Beck and Cynthia Andres. *Extreme Programming explained: embrace change*. Addison-Wesley, Boston, 2. ed. edition, 2005.

[4] Dan Cumbo, Earl Kline, and Matthew S. Bumgardner. Benchmarking performance measurement and lean manufacturing in the rough mill. *Forest Products Journal*, 56(6):25 – 30, 2006.

[5] Lars-Ola Damm. *Early and Cost-Effective Software Fault Detection*. PhD thesis, Blekinge Institute of Technology Doctoral Dissertation Series No. 2007:09, 2006.

[6] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006.

[7] Mark Denne and Jane Clelund-Huang. *Software by the numbers: low-risk, high-return development)*. Prentice Hall, 2004.

[8] Paolo Donzelli and Giuseppe Iazeolla. A software process simulator for software product and process improvement. In *Proceedings of the International Conference on Product Focused Software Process Improvement (PROFES 1999)*, pages 525–538, 1999.

[9] Kathleen M. Eisenhardt. Building theories from case study research. *Academy of Management Review*, 14(4):532–550, 1989.

[10] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A model for technology transfer in practice. *IEEE Software*, 23(6):88–95, 2006.

[11] Tony Gorschek and Claes Wohlin. Requirements abstraction model. *Requir. Eng.*, 11(1):79–101, 2006.

[12] John M. Gross and Kenneth R. McInnis. *Kanban made simple: demystifying and applying Toyota's legendary manufacturing process*. AMACOM, New York, 2003.

[13] Alan R Hevner, March T. Salvatore, Jinsoo Park, and Ram Sudha. Design science in information systems research. *MIS Quarterly*, 28(1):75–103, 2004.

[14] Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, and Christian Nyberg. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. *Journal of Systems and Software*, 59(3):323–332, 2001.

[15] L. Kemayel, Ali Mili, and I. Ouederni. Controllable factors for programmer productivity: A statistical study. *Journal of Systems and Software*, 16(2):151–163, 1991.

[16] Craig Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley, Boston, 2004.

[17] Brian Maskell and Bruce Baggaley. *Practical lean accounting: a proven system for measuring and managing the lean enterprise*. Productivity Press, 2004.

[18] Peter Middleton. Lean software development: Two case studies. *Software Quality Journal*, 9(4):241–252, 2001.

[19] Peter Middleton, Amy Flaxel, and Ammon Cookson. Lean software management case study: Timberline inc. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005)*, pages 1–9, 2005.

[20] Douglas C. Montgomery and George C. Runger. *Applied Statistics and Probability for Engineers*. Wiley, 2006.

[21] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead-time reduction in a software product customization process with value-stream maps. In *Proceedings of the 10th Australian Conference on Software Engineering (ASWEC 2010)*, 2010.

[22] Emma Parnell-Klabo. Introducing lean principles with agile practices at a fortune 500 company. In *Proceedings of the AGILE Conference (AGILE 2006)*, pages 232–242, 2006.

[23] Richard T. Pascale. *Managing on the edge: how the smartest companies use conflict to stay ahead*. Simon and Schuster, New York, 1990.

[24] G.I.U.S. Perera and M.S.D. Fernando. Enhanced agile software development hybrid paradigm with lean practice. In *Proceedings of the International Conference on Industrial and Information Systems (ICIIS 2007)*, pages 239–244, 2007.

[25] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, 82(9):1479–1490, 2009.

[26] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404, 2010.

[27] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit (The Agile Software Development Series)*. Addison-Wesley Professional, 2003.

[28] David Raffo. Evaluating the impact of process improvements quantitatively using process modeling. In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative research (CASCON 1993)*, pages 290–313. IBM Press, 1993.

[29] Donald G Reinertsen. *Managing the design factory: a product developers toolkit*. Free, New York, 1997.

[30] Colin Robson. *Real world research: a resource for social scientists and practitioner-researchers*. Blackwell, Oxford, 2. ed. edition, 2002.

[31] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[32] Bridget Somekh. *Action research: a methodology for change and development*. Open University Press, Maidenhead, 2006.

[33] Brian White. *Software configuration management strategies and Rational ClearCase: a practical introduction*. Addison-Wesley, Harlow, 2000.

[34] Colin Willcock, Thomas Deiss, Stephan Tobies, Stefan Keil, Federico Engler, and Stephan Schulz. *An Introduction to TTCN-3*. John Wiley & Sons Ltd., 2005.

[35] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in Software Engineering: An Introduction (International Series in Software Engineering)*. Springer, 2000.

[36] James P. Womack and Daniel T. Jones. *Lean thinking: banish waste and create wealth in your corporation*. Free Press Business, London, 2003.

[37] Robert K. Yin. *Case study research: design and methods*. Sage Publications, 3 ed. edition, 2003.

Chapter 9

Lean Software Maintenance

Kai Petersen
Submitted to a conference

9.1 Introduction

Development activities are distinguished into initial software development where the software product is specified, implemented and tested and thereafter delivered to the market, and software maintenance [1]. Software maintenance distinguishes between enhancements (i.e. adaptations to the delivered software to provide minor functionality enhancements, optimize resources and performance, etc.) and corrections (i.e. resolving problems reported by customers, such as defects). Software maintenance is a large part of the software development effort, with effort for maintenance being in the range of 50-80 % of all software development effort [18]. Given these observations software maintenance is an essential activity in software development and hence improvements in this activity have the potential to increase the efficiency and effectiveness of an overall software development organization.

Lean product development has revolutionized the way products are built by identifying waste and providing analysis tools for the production process to make it more efficient and effective [9]. To leverage on the benefits achieved in lean product development (high quality, quick response to customer needs, just in time development with little work in progress) lean has become popular for software development as well [13, 14, 15]. Lean and agile software development share a number of principles

and practices, such as self-organizing teams, sustainable pace, face-to-face conversation, the focus on technical excellence, and so forth. In comparison to agile lean has a stronger focus on providing principles and practices that should help in systematically analyzing and improving a process to become highly effective and efficient, such as "See the whole", "Avoidance of waste", "Short cycle times and rapid feedback", and "Build in quality" (see Chapter 2). The principles are supported by lean tools, the best known being value stream maps [10], cumulative flow diagrams [3] and Kanban [13, 3]. The analysis of the related work showed that no solution is available to analyze software maintenance from a lean software development perspective.

The contribution of this paper is to provide a novel analysis approach for the maintenance process to realize the lean principles mentioned above. "See the whole" is supported by focusing on multiple dimensions (i.e. product quality, continuous delivery of valuable results, process cycle-times and iterations due to internal quality issues) [1]. For the analysis of each dimension's measures and visualizations are provided. For the identification of the measures and visualizations the Goal Question Metric (GQM) approach [4] was used as guidance in identifying a solution supporting the lean principles. The goal driving the analysis was to *"Increase the efficiency and effectiveness of software maintenance from a lean software development perspective"*. The proposed solution is an extension of the software process improvement through lean measurement (SPI-LEAM) method (see Chapter 7) which analyzes how an overall software development process performs considering work in process in relation to capacity. The goal of the method is to identify signals for problems in different development disciplines (main requirements flow, change requests, and customizations, software testing, and maintenance). Hence, the method provides a high level overview of the different disciplines together. In order to identify problems on the more detailed level a drill-down of each dimension is needed to gain a more in-depth understanding of the underlying behavior. The approach presented in this paper is the drill-down analysis that is specific for the maintenance process. We would like to point out that the solution supports SPI-LEAM, but can also be implemented independently of that solution.

The solution has been evaluated in an industrial case study at Ericsson AB, located in Sweden. For that the solution was applied on the maintenance requests (MRs) entering the studied company site in the year 2009 as those reflect the performance of the maintenance process currently applied at the company. The goal of the case study is to demonstrate the ability of the approach to show the absence or presence of ineffectiveness and inefficiency in the studied process.

The remainder of the paper is structured as follows: Section 9.2 presents the related work and based on that demonstrates the need for the proposed solution. Section 9.3 provides the solution for lean software maintenance. Section 9.4 presents the research method to evaluate the solution. Section 9.5 illustrates the results obtained. Thereafter,

Section 9.6 discusses the results and Section 9.7 concludes the paper.

9.2 Related Work

The related work focuses on measurements proposed in the maintenance context to assess maintenance performance in different ways. To get a good idea of the literature in the topic area the titles and abstracts of the International Conference on Software Maintenance have been browsed manually as this is the primary forum for maintenance research. Furthermore, we searched Inspec and Compendex as well as Google Scholar, given that they cover a variety of databases, such as IEEE Xplore, Wiley Interscience, Elsevier, and ACM. To identify the literature we searched for: "Software Maintenance" AND (Measurement OR Metric OR Measure OR Quantify OR Productivity OR Efficiency). The lists were browsed and the selection of articles included in the related work were made based on the titles. Software maintenance in combination with lean has, to the best of our knowledge, not been researched before. Hence, no related work for that combination is included.

Alam et al. [2] observed that progress in maintenance is often captured through manual collection of metrics or through automatically collected metrics, such as lines of code. As an improvement to measuring progress as an important performance measure they propose to record code changes and study the time dependencies between them. For example, if a class uses a function of another class then there is a time dependency between them as the used class has to be implemented first. The progress tracking is an analogy to construction where a building is continuously built based on dependent changes. This allows to analyze whether new MRs are built upon new changes done recently, or old changes, or whether they are independent. To demonstrate the approach it was applied on two open source systems. A difference could be identified, e.g. it shows that in one system the majority of maintenance activities were based on new changes, while in another system the number of changes built on new and old changes as well as independent changes was similar. One limitation of the method is that it does not show the progress with regard to the backlog of changes to be implemented.

Schneidewind [17] studied enhancement projects and provided a number of performance measures. The performance measures were mean time between failure when testing the enhancement (calculated as the ratio of test execution time and the total number of failures), the defect density with regard to the changes in the code counted as changed lines of code, and the overall test time needed for the testing of the enhancement implementation. Overall, the primary focus of the measurements was the product.

Henry et al. [7] provide a process modeling approach for maintenance with metrics connected to it. The model is divided into different abstraction levels (activities, tasks, and procedures), where procedures are a refinement of tasks, and tasks are a refinement of activities. At the different abstraction levels measures were proposed on process and product level. On process level the following measures should be collected with regard to the abstraction levels:

- Process (activities): Effort expanded and progress towards a predefined milestones.
- Process (tasks and procedures): Completion rate of tasks and defects corrected per week.

The measures on product level were also divided into abstraction levels for the measures (product level, module level, and code level). The following measures should be collected according to the article:

- Product (product level): Number of upgrades to be implemented, defects discovered, uncorrected defects, priority of defects, priority of defects.
- Product (module level): Impact of upgrade changes on elementary parts of the system.
- Product (code level): Number of LOC added, changed, or deleted.

The article also raised the importance of visually presenting the data to management, which aids communication. However, no visualization examples were provided. In addition the authors recommend the usage of statistical analysis to understand how individual measures are connected, e.g. by calculating correlations. As pointed out in the study a large number of measures have to be collected.

Sneed and Brössler [19] proposed a set of measurements to evaluate software maintenance performance. The first measurement was related to productivity measured as the output (size * change rate) divided by the input (maintenance effort). The productivity measure is complemented by a quality measure of defect density, calculated as number of defects per lines of code (old code and changed code). As there is a challenge of keeping different software artifacts consistent degradation also needs to be captured, e.g. when a document does not conform to system implementation then the the author talks about document degradation. Another important aspect to be captured is the user perception of the success of the maintenance task. Therefore, the author recommends to ask users for different criteria to be rated on a Likert scale (e.g. performance, time for serving the MR, etc.). User satisfaction is measured as the ratio of points achieved divided by the total points possible to achieve.

Rombach and Ulery [16] used the Goal Question Metric paradigm (from hereon referred to as GQM) to identify goals, questions and metrics for analyzing the maintenance process. In GQM, the first step is to identify the goals to be achieved, which drive the identification of questions to be answered in order to achieve the goals. In order to answer the questions a number of metrics are identified. That is, the GQM provides a top-down approach for the goal-driven identification of measures. The goals identified were the ability to compare maintenance activities between two programming languages, and to collect measures about the behavior of maintenance for predictive purposes. The measures collected were the number of changed module per maintenance task, effort (staff-hours) needed for isolating what to change (i.e. impact analysis), effort (staff-hours) for change implementation, and portion of reused documentation per maintenance task. As pointed out in the article the overall implementation of the measurement program took about six staff-years. However, the authors pointed out that the long-term benefits will likely outweigh the investments made.

Stark [20] provided an experience report on the introduction of a measurement program for software maintenance. The measures were also derived based on the GQM paradigm. The GQM led to three categories of measures, namely measures regarding customer satisfaction, costs of the maintenance program, and schedule. The measures capturing customer satisfaction was the backlog of MRs to be fulfilled, the cycle time, and reliability. The cost category included measures for cost per delivery of maintenance result in dollar, and cost per maintenance activity. In addition the maintenance effort of staff and the number of changes needed per request were captured, as well as the effort spent on invalid change requests. The schedule category was captured by measuring the difficulty of implementing the change (e.g. if a change is very complex it takes longer time). Additional measures were the number of changes planned for delivery, and the number of changes meeting the schedule.

Chan [5] raises the importance of addressing MRs quickly, i.e. with short lead-time. To capture the lead-time he distinguishes queuing time and service time. Queuing time is the time from arrival of a request until the actual implementation and servicing of the request is started. In other words, queuing time is waiting time. Service time is the time where actual work is done on the request including implementation and verification.

The following observations regarding the related work can be made: Often the literature reported measures that are relevant for lean software development, but they are reported in isolation. For example, it is important to distinguish waiting time and service time (cf. [5]) as waiting time is often much easier to improve than actual work practices. In addition, the measures do not make use of visualizations considered important in a lean context to communicate and easily identify improvement potentials. Another important aspect is that some approaches require considerable effort in implementation. The approach introduced by Henry et al. [7] required extensive modeling

and hence is a challenge to implement. Furthermore, Rombach and Ulery [16] reported that the introduction of their maintenance measurement program required considerable effort.

Given the benefits achieved in lean product development the novel contribution of the approach presented in this paper is to:

- Provide an analysis approach that allows the analysis of the maintenance process driven by lean principles, as in previous related work some lean measures have been proposed in isolation, but were not captured in one approach. Furthermore, the lean software maintenance approach captures multiple dimensions (i.e. product and process).
- Make use of visualizations for easy communication with management, as very few approaches provide good guidelines of how to visualize collected measures.
- Base the analysis tool on as few needed measurements as possible to allow for an easy introduction in software organizations.

9.3 Lean Software Maintenance

As mentioned earlier, the goal driving the development of lean software maintenance is to increase the efficiency and effectiveness of software maintenance from a lean software development perspective. Hence, the questions asked are linked to the lean principles mentioned in the introduction. The following GQM questions were identified:

- *Q1: How is the quality over time delivered to the customer?* This question is linked to the goal as it relates to the concept of building quality in, i.e. knowing the quality level over time provides an indication of whether an organization succeeded in that principle. Hence, when the answer is that good quality is delivered then the performance is good from that lean perspective.
- *Q2: How continuous and with what throughput do we process MRs from customers?* This principle relates to waste, and in particular work in progress. Work in progress is not yet delivered to the customer and hence is considered waste. This type of waste motivated just in time which is the production without inventories in between. In other words, it is important to not have half finished work-products being stuck in the process for too long as there is a risk that they become obsolete, wasting the already invested effort. Continuous means that work products should be produced all the time instead of queuing them up and

realizing them all at once. Queuing up and building inventories often leads to the risk of an overload situation [12]. In addition, queuing up delays feedback and hence it becomes harder to fix problems. High throughput refers to velocity, i.e. it is desirable to resolve many MRs per time unit. Overall, the answer to this question relates to the goal as much waste in the process indicates inefficiencies (cf. [13].

- *Q3: How fast do we respond to the needs raised by the customer?* This question captures the ability of the organization of how fast it is able to resolve a request from its emergence to delivery. The fast delivery of valuable software is considered as an important aspect of efficiency in lean [12] as well as agile software development [8].

- *Q4: What is the level of work-load?* It is important to understand the work-load level on the developers as an overload situation can hinder the continuous flow and throughput, as well as increase lead-times [12]. In Chapter 7 we provided the analogy of a highway. If the highway is almost full then the traffic goes slowly and is close to a stand still. The best flow can be achieved with a work load below capacity allowing the developers to think about solutions and to correct mistakes made.

The following measures are proposed for answering the GQM questions:

- *M1 for Q1:* Maintenance inflow of requests per week (Section 9.3.1).

- *M2 for Q2:* Visualization of the flow of MRs through the maintenance process in the form of cumulative flow diagrams (Section 9.3.2).

- *M3 for Q3:* Lead-time measurement considering queuing/waiting time and productive/value adding time (Section 9.3.3).

- *M4 for Q4:* Statistical process control chart showing stability of work-load over time (Section 9.3.4).

The measures mainly concern efficiency (i.e. doing things right to produce MRs in a continuous manner with high throughput and short lead-time). To incorporate the effectiveness (doing the right thing) we need to incorporate the importance and severity of the MRs and based on that conduct separate analyses for each of the questions. This allows to compare the performance of maintenance tasks on critical versus less critical MRs. The right thing to do in this case is to have better performance for the more critical tasks.

9.3.1 Maintenance Inflow (M1)

The maintenance inflow shows the incoming requests from customers for needed maintenance activities. As pointed out earlier, the MRs can be either corrective or enhancements. The inflow should be measured as number of new MRs per time unit (e.g. week). The measure can be visualized through a control chart, showing the time on the x-axis and the number of new MRs on the y-axis (also referred to as a time-series). To detect whether the process is under control the chart could be extended by marking the mean value in the chart, and plotting the upper and lower control limits (usually two or three standard deviations away from the mean).

9.3.2 Visualization Through Cumulative Flow Diagrams (M2)

A fictional example of the construction of a cumulative flow diagram is shown in Figure 9.1. The x-axis shows the time-line and the y-axis shows the cumulative number of of MRs. The top-line (marked as inflow by the arrow) is the total number of MRs in development. In week 9 this was around 160, while it increased to around 220 in week 20. Even though the flow is shown we propose to treat it as a separate measure (M1) and with that analyze the stability of the process with regard to the mean value and the deviations from the mean through upper control and lower control limits, which is not possible within the flow diagram.

The second line from the top represents the hand-over from phase A to phase B, the following line from phase B to phase C, and so forth. The vertical distance between two lines shows the work in process for a phase at a specific point in time. For example, in week 15 there are about 50 MRs in Phase B.

From the flow diagram a number of interpretations can be made with regard to the continuous flow and the high throughput.

- *Continuous flow:* Observing the figure it is visible that the flow of hand-overs from phase C to phase D is discontinuous. That is, there is a long time of inactivity (week 9-13) and then suddenly a large hand-over occurs. That means, for example, that work done in week 9 and 10 can receive feedback from the following phase 5 weeks later. In addition, a long time of inactivity might lead to an overload situation when the work has to be done at once, and at the same time has potential of causing quality problems. For example, if phase D would be an integration testing phase with long times of inactivity a big-bang integration would become necessary.

- *Throughput:* The throughput is characterized by the rate in which MRs are handed over from one phase to the other. As can be seen the hand-overs from

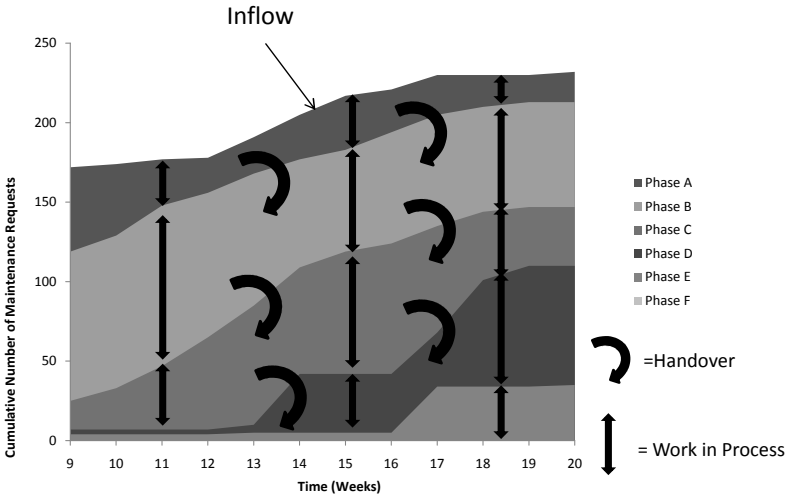


Figure 9.1: Cumulative Flow Diagram

phase A to B had less throughput from week 9 to 13 than the hand-overs from phase B to C, while the hand-overs became very similar after week 14. The throughput allows to make two observations: If the hand-over in a phase i is higher than in the phase $i+1$ this indicates a bottleneck as the work tasks come in with a higher pace than they go out. The other way around would indicate that the phase is running out of work, which in the case of maintenance could be good as it means free resources for other tasks in new software development.

The analysis can be done for single phases (e.g. analysis of MRs), or to get a picture of the overall performance of maintenance from start to finish, the incoming rate could be compared with the rate in which MRs are finalized.

9.3.3 Lead-time measurement (M3)

In order to measure the lead-times we propose to follow MRs through the maintenance flow by assigning states to them. A theoretical illustration of this is provided in Figure 9.2. The figure shows different activities of the maintenance flow. When a certain activity related to the MR is executed (e.g. registration of request, analysis of request, and so forth) then the request is put into that state. The lead-time is determined by keeping track of the duration the requests resided in different states related to activities.

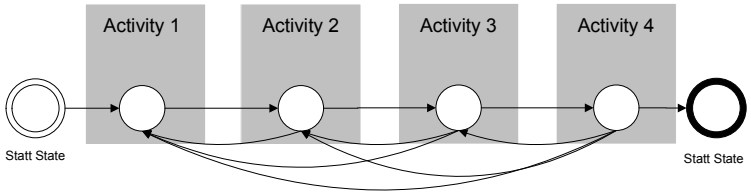


Figure 9.2: Measuring Lead-Time

In order to be able to measure the lead-time a time-stamp has to be captured whenever the MR enters the state, and leaves the state. Depending on the maintenance flow a request can go back to a state it already passed (e.g. due to iteration as the request was rejected by regression testing). With this approach a number of different lead-time measures can be conducted:

- LT_a: Lead-time of a specific activity a based on the duration a request resided in the state related to the activity.
- FC: The number of feedback-cycles allows to see how often a request entered the same activity. This is, for example, interesting when a request is rejected from testing indicating quality problems as a reason for prolonged lead-times.
- LT_{n-a}: Lead-time starting with an activity a and ending with an activity n. In order to calculate this lead-time, the sum of the durations of all activities has to be calculated. Thus, a company can calculate the overall lead-time of the whole maintenance life-cycle, or specific sets of activities within the life-cycle.
- WT: The measurement can also easily incorporate waiting times. Waiting times are considered a waste in software development [13, 14, 15]. They can be captured by not only having states for identified activities assigned to MRs, but also having states for waiting times in between activities. Thus, when an activity on a MR is completed it needs to change into the state waiting. When the next activity picks up the MR, then it leaves the state waiting. This information also provides to create value stream maps which is a lean analysis tool specifically targeted towards identifying waiting and queuing times and showing them visually [10].

When collecting the lead-time measures the mean values and variances should be analyzed and compared with each other. For example, one should compare the lead-time of critical and less critical MRs. To visualize the differences we propose the use of box-plots showing the spread of the data.

9.3.4 Work-load (M4)

The work-load analysis is interesting with regard to the work-load in value-adding activities over time. For that purpose the work in progress measured as the number of MRs at a specific point in time should be plotted and analyzed using statistical process control. The control chart then should be used to have a dialog with the developers executing the activities to determine which work level is considered an overload situation. In order to be able to properly analyze the workload the requests should be estimated based on their complexity, as a complex problem is likely to be causing more work-load than an easy to fix problem. This can be, for example, done by categorizing requests in very complex, complex, and less complex. This can mean different things for different companies, and hence each company has to define its own thresholds for the categorization.

9.3.5 Prerequisites

The analysis for M1 to M4 can be realized with relatively few measurement points. The following information need to be captured to conduct the most basic analysis:

1. Registration of MRs with time-stamps to determine the in-flow (realizing M1).

2. A tracking system for states of the maintenance process. When the state changes it has to be updated in the system. With the update the system keeps track of the date and time when the update was made (realizing M2, M3, and M4).

3. A weighting of the MR with regard to value and complexity. The weighting supports the analysis of effectiveness by comparing the performance in resolving critical verses less critical MRs.

In the measurement it is of benefit to also have states for waiting times as well, as has been argued for previously (see Section 9.3.3).

9.4 Research Method

The research method used is an industrial case study [22], the study allowing to understand a specific situation (the use of lean software maintenance) in a given context. The case study can be characterized based on the template proposed in Wohlin et al [21].

- Analyze *Lean Software Maintenance* for the purpose of *evaluation*,

- with respect to *the ability to show the presence/absence of inefficiencies and ineffectiveness*,
- from the point of view of *the researcher*,
- in the context of *large scale industrial software development dealing with corrective maintenance*.

The description of the case study illustrates the main components as described in Yin [22], namely the case and context, the units of analysis, propositions, and the data collection and analysis. In addition, threats to validity are discussed.

9.4.1 Case and Context

The case being studied is a development site of Ericsson AB, a Fortune 500 company working with large-scale software development producing telecommunication and multimedia applications. It is important to describe different dimensions of the context (e.g. product, market, process) in order to judge the outcome of a study and to be able to generalize the results [11]. Hence, Table 9.1 shows the context elements for this study. It is important to note that the company is dealing with business critical applications with MRs on performance and reliability. The products are developed for a market, meaning that it is not developed for one specific customer (bespoke development). Instead, the product is offered to a market with many potential customers, not knowing exactly in advance who will actually buy the product.

9.4.2 Unit of Analysis

The unit of analysis is the maintenance process used for maintaining one large system developed at the case company. Figure 9.3 provides an overview of the maintenance process. The process starts with a customer raising a MR, which then is registered in the system by support. In the next step the MR is moved to the appropriate design organization. The next step is the analysis of the MR to understand it. In addition test cases are designed and executed to verify the MR. If the MR is understood and the test cases are clear it goes to the design, implementation, and test phase. If the MR is not clear, further analysis is necessary. In the analysis the design organization is working together with support receiving the information about the problem from the customer, and with experts knowing the system very well who serve as a consultant. When the MR is understood it is designed and implemented. The implementation of the MR needs to be verified, the verification being confirmed in a so-called technical answer, confirming that the solution is complete, coding standards are met, and that

Table 9.1: Context Elements

Element	Description
Product Size	More than 850,000 Lines of code
Quality	Business critical application with strict requirements for performance and reliability.
Certification	ISO 9001:2000
Requirements engineering	Market-driven process.
Testing practices and tools	Application and integration test verifying if components work together (JUnit, TTCN3), regression test for maintenance with (TTCN3).
Defect tracking	Company-proprietary tool allowing to track MRs.
Team-size	6-7 team members.
Distribution	Globally operating company.

regression tests have been passed. If this is not the case the MR re-enters the analysis and/ or design and implementation stage. If the MR has passed it goes either to a correction package which is further tested as a whole and then released with a number of correction, or in some cases it can go directly to the customer.

Based on the process described above the following states are tracked within the company's tool: (1) MRs waiting for registration in design, (2) MRs waiting for start of analysis, (3) MRs in analysis and implementation; (4) MRs for which a solution has been proposed; (5) MRs waiting for a technical answer confirming successful implementation; (5) MRs waiting for finalization; and (6) finished MRs. A loop is modeled in the tool for the situation where the technical answer does not accept the solution and thus the analysis and/ or implementation has to be redone. The number of iterations are numbered as revisions, revision A being through in first iteration, B being through in second iteration, and so forth.

9.4.3 Proposition

A proposition is similar to a hypotheses and states the expected outcome that is either supported or contradicted by the findings of the study [22]. The following proposition is stated for this study: *Measures allow to capture the presence or absence of ineffi-*

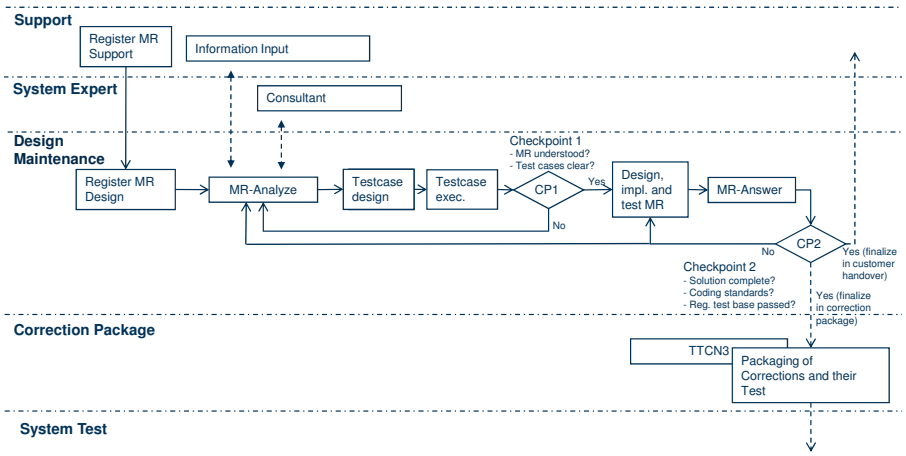


Figure 9.3: Maintenance Process

ciencies and ineffectiveness with regard to the questions raised and allow to discover the need for improvements.

9.4.4 Data Collection and Analysis

The data is collected through the company proprietary tool for keeping track of MRs that were internally or externally reported. The system fulfilled the prerequisites stated in Section 9.3.5 to a large extent. That is, the incoming MRs are registered with their source and it is also visible which person entered them into the system. The time of entry is kept track of. In addition, the process steps are represented as states mirroring the process shown in Figure 9.3. This allows for the drawing of the cumulative flow diagrams and the measurement of the lead-times as defined by the lean software maintenance solution presented in this paper. Furthermore, the MRs are classified (weighted) based on their importance into as A, B, and C. The information about the classification has been obtained by a person having worked in testing and with experience in maintenance.

- *A:* MRs that concern performance or stability of the system are mostly classified as A. They are important and most of the time they are not easy to fix.
- *B:* Problems in this category often concern the robustness of the system. In some cases robustness problems are also classified as A.

- *C:* These problems are less severe and more easy to fix, such as fixes in single functions. Depending on how intensively the function is used by the user, or how hard the functional MR is to correct the MR can in cases also be classified as B.

For the analysis the MRs of the year 2009 were used as the performance measure, reflecting the current practice in conducting maintenance activities at the company. The analysis was done by applying the lean software maintenance solution on the data and conduct an analysis to demonstrate whether the solution is able to show the presence or absence of inefficiencies and ineffectiveness. The interpretation was done by the researcher who has good knowledge a the company's processes as he is embedded in the company. In addition, the results have been presented to a fellow practitioner to check whether the practitioner agrees with the observations made by the researcher.

9.4.5 Validity Threats

Validity threats are important to discuss to support the interpretation of the data by others. In the following the validity threats and their relevance for this study are discussed.

Correctness of data: One threat to validity when working with industrial data is the correctness and the completeness of the data. In the case of the company the tracking system for MRs has been used for almost 15 years and hence the practitioners are very familiar with using the system in their work, which reduces the threat of incorrect data. When changing the state of a MR the system automatically keeps track of the dates, which avoids that dates could be wrongly entered. Hence, the correctness and completeness of the data is given.

Company specific maintenance process: The maintenance process of the company is specific for the company. However, the method should be generally applicable to other maintenance processes as well if the explained prerequisites are met in the company. Another limitation for generalization is that a corrective maintenance process has been studied. However, the principle solution proposed in this paper stays the same, just different phases and thus states have to be identified and need to be kept track off.

One company: When studying a company the results are true in the context of the company. In order to aid in the generalization of the results the context has been described. That is, the results were observed in a market-driven context working with large-scale software development.

Interpretation by the researcher: Another threat to validity is the correct interpretation of the data by the researcher. This threat was reduced as the researcher is embedded in the company and hence has knowledge about the processes. A bias in the analysis was reduced by presenting the results of the analysis to a practitioner working with testing and maintenance at the company. As the practitioner agreed with the

interpretation this threat is considered under control.

9.5 Results

9.5.1 Maintenance Inflow (M1)

Figure 9.4 shows the inflow of A, B, and C defects over time. It is clearly visible that B-faults occur most often and continuously with a few peaks. A-faults appear rather randomly and are spread around, which would be expected and desired as if many would be reported at once a disturbance of the regular development process can be expected. C-faults are less frequent. Overall, the result shows that it would be

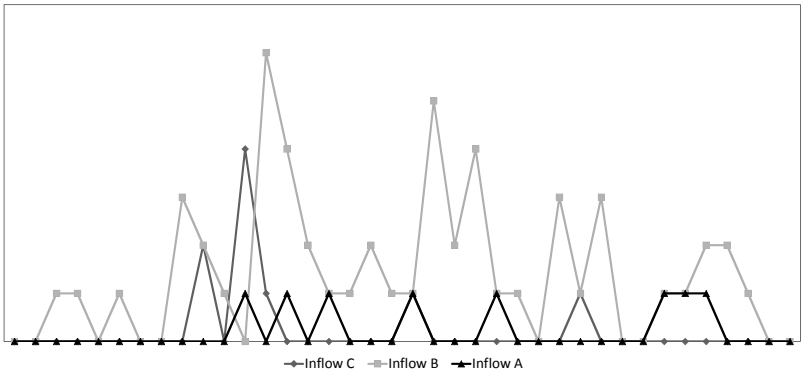


Figure 9.4: MR Inflow for A, B, and C MRs (x-axis shows time and y-axis number of MRs)

worthwhile to investigate a reason for the peaks when many faults are reported together. Otherwise, no significant inefficiencies or a particularly poor performance with regard to A-MRs can be observed.

9.5.2 Visualization Through Cumulative Flow Diagrams (M2)

Figure 9.5 shows the flow of A-faults. It is apparent that the actual analysis and implementation appears to be a bottleneck in the beginning, leading to a high amount of the MRs being proposed as a solution at once. In addition the area of MRs waiting for finalization shows that improvements would be beneficial, as for a long time none

of the accepted MRs are finalized and thus become available for the customer or for inclusion in a correction package.

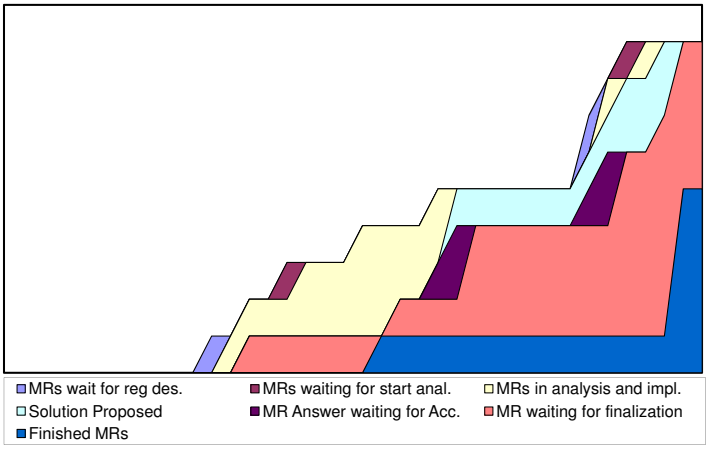


Figure 9.5: Maintenance Process Flow A MRs

The maintenance flow for B-MRs (Figure 9.6) shows that MRs are analyzed and implemented continuously. The same observation as for the A-MRs can be made here as well, with the difference that the B-MRs are finalized more continuously. However, it is apparent that the rate in which MRs (solution proposals) are accepted is much higher than the rate of finalization.

For the C-MRs a similar observation can be made, i.e. the MRs should be finalized in a more continuous manner (see Figure 9.7).

With regard to the iterations needed to successfully pass a MR through internal quality assurance Figure 9.8 illustrates that over 70 % of the MRs make it the first time. Overall, the analysis shows that no specific inefficiencies can be detected here. Of course, it would be worthwhile to investigate the reasons of why some MRs require several revisions (e.g. MRs related to revision C and D).

With regard to differences between the A, B, and C MRs it should be noted that it is particularly important to avoid the queuing of MRs waiting for finalization in the A-case. The B-case appears to be more continuous, but still shows a bottleneck in this phase.

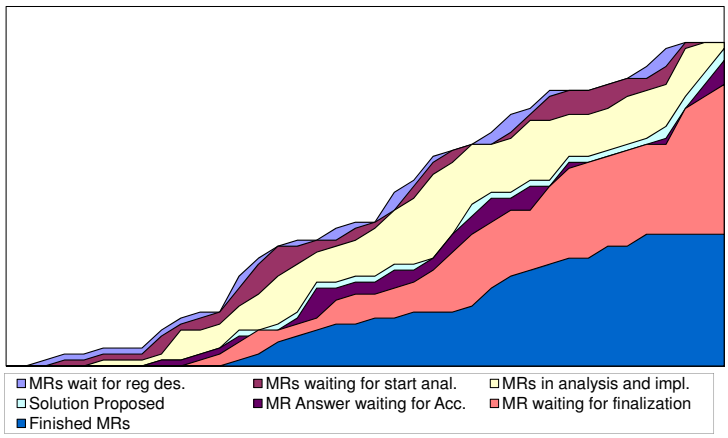


Figure 9.6: Maintenance Process Flow B MRs

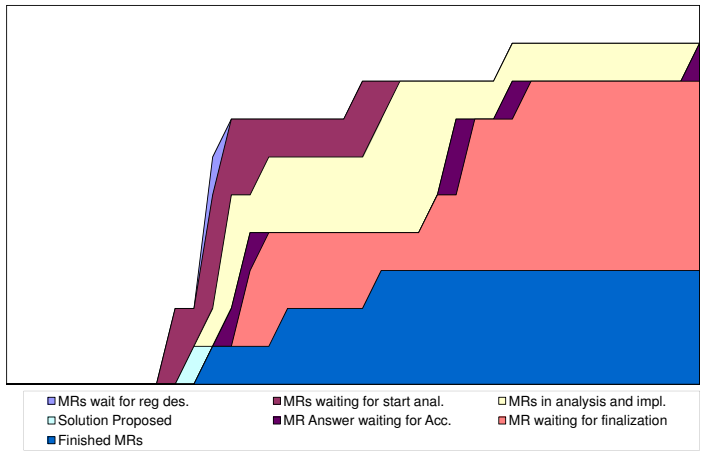


Figure 9.7: Maintenance Process Flow C MRs

9.5.3 Lead-Time Measurement (M3)

Figure 9.9 shows the lead-time of how long MRs reside in the different states, namely MR waiting for registration in design (S01), MR waiting for the analysis to start (S02),

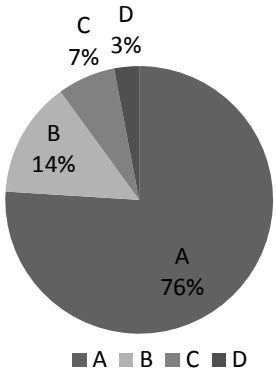


Figure 9.8: Revisions

MR in analysis and implementation processing (S03), MR waiting for an answer acknowledging the solution (S04), and MR waiting for finalization (either as a direct delivery to the customer or the packaging into a maintenance request) (S05). The total lead-time is shown as well.

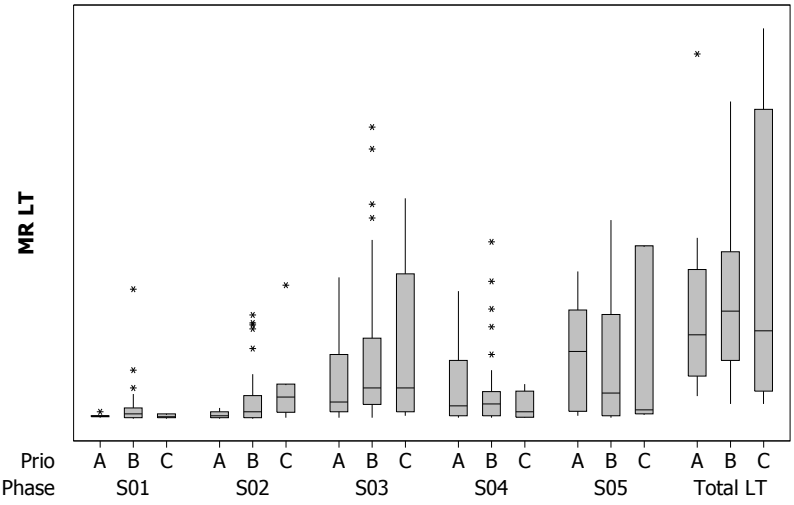


Figure 9.9: Leadtime

Comparing A, B, and C MRs it is apparent that A and B MRs have a high overlap of the plots, and that the median values are similar. One could say that the lead-time should be the shortest for A-MRs. However, as noted before A-MRs are hard to fix and often rooted in performance problems, thus the similar lead-time in the analysis and design phase could be justified. However, at the same time it is striking that long lead-times are observed with regard to waiting times, the most significant waiting times being in waiting for finalization. In fact, the waiting times are very similar to the value adding time where the actual analysis and implementation takes place. As waiting time is often easier to improve in comparison to productive time the figures show an improvement potential. The total lead-time shows that MRs classified as C have a similar median value as MRs classified as A, but the upper quartile for the lead-time is much higher. This is an indication for the lower priority of C MRs, and thus is an indication that the company focuses on effectiveness in concentrating more on getting A and B MRs to the market quickly.

9.5.4 Work-Load (M4)

The workload is illustrated as individual values and moving ranges in Figure 9.10. The continuous middle line shows the mean value, while the dashed lines show the upper and lower control limits being three standard deviations away from the mean. If values are outside of the control limits the situation is considered out of control. In this case a peak work-load can be seen in the middle of the graph. For management to gain a better understanding of the workload situation we propose to use the chart and discuss the workload situation with the developers. This allows to determine how much workload should be in the process at any given time to not overload the development organization.

9.6 Discussion

The proposition of the study stated that *the proposed solution allows capturing the presence or absence of inefficiencies and ineffectiveness with regard to the questions raised and allow to discover the need for improvements.* Confirming the proposition indicates the usefulness of the method. In the results the method was used to show the presence or absence of inefficiencies and ineffectiveness. The following was identified:

- With regard to the inflow of MRs into the development organization no striking quality issues have been identified with regard to A-defects, they appeared randomly and did not occur in large batches. With regard to B-defects we have

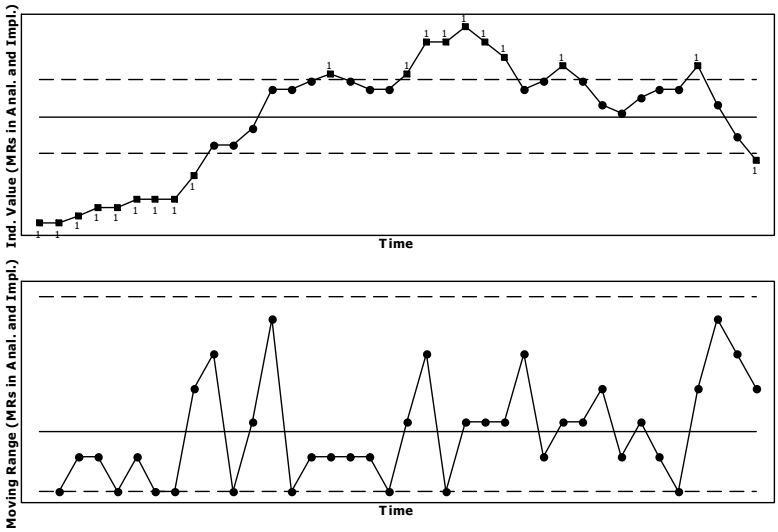


Figure 9.10: Workload

shown that some peaks were visible, which should be investigated. Hence, the method showed some potential inefficiencies here. One way of investigating the faults is to define a test strategy determining when the fault should have been found (known as fault-slip through [6]). This allows to know how early the fault could have been detected.

- The analysis of the flow showed that a bottleneck exists in finalizing the MRs across all types of MRs (i.e. A, B, and C). Hence, the reason for this waiting time should be identified with priority on the most critical MRs. It is interesting to observe that the bottleneck appeared in a phase which is regarded as waiting time, which means that it could be more easily improved. No particular inefficiencies were identified with regard to iterations needed to pass through the internal quality control.

- A comparison of the lead-times showed that more than 50 % of the lead-time appears to be waiting time. This is an interesting result as waiting time can more easily reduced than productive time, meaning that the measures show potential for the organization to significantly shorten their response time to MRs.

- It was also demonstrated that peaks of work-load could be identified.

In order to get a holistic picture it is important to bring the results together, as is done in Figure 9.11. The figure shows the presence of efficiencies and effectiveness on the top, and the discovery of inefficiencies and ineffectiveness on the bottom. Inefficiencies and Ineffectiveness are to be discovered as they show the improvement potential in the process. The efficiency generally refer to the performance that could be improved. The effectiveness shows strength and improvement potential with a focus on a comparative analysis between A, B, and C MRs. Overall, this analysis that the proposition stated for this study holds, i.e. lean software maintenance is able to show the presence or absence of inefficiencies and ineffectiveness.

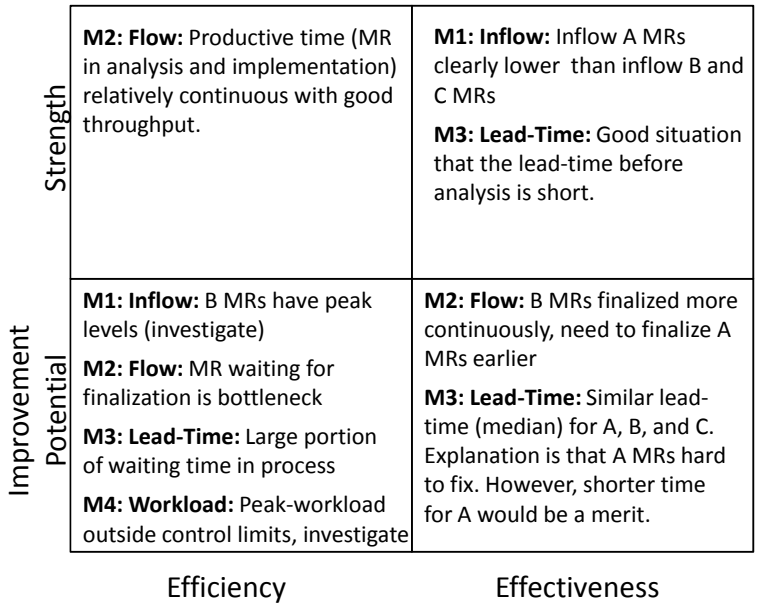


Figure 9.11: Efficiency and Effectiveness Analysis

One important limitation of the approach as implemented at the company was identified. The classification of A, B, and C fault should clearly distinguish between criticality (how important is the MR for the customer) and complexity (how hard is it to fix the MR). This information allows to analyze which MRs should receive primary focus. For example, MRs with high priority that are easy to fix should be realized first.

9.7 Conclusion

In this paper a novel approach of implementing lean software maintenance has been introduced. The solution relies on the analysis of the software maintenance process with a specific focus on lean aspects (see the whole, build quality in, continuous delivery of value, etc.). The goal of the approach was to identify inefficiencies and ineffectiveness with regard lean principles. Four analysis tools have been proposed, namely the inflow of MRs, the visualization of the flow through the maintenance process with cumulative flow diagrams, lead-time measures based on state diagrams, and the analysis of workload peaks with process control charts.

The approach has been evaluated in an industrial case study at Ericsson AB. The study demonstrated that the approach was able to identify the presence or absence of inefficiencies and ineffectiveness in the maintenance process. We also have shown that lean software maintenance requires the company to keep track of few measurements, still allowing for a comprehensive analysis. In fact, the system implemented at the company allowed the immediate application. The prerequisites for implementing the approach are quite minimal, a company has only to keep track of registration of MRs with time-stamps, state-changes of the MRs in the process, and the criticality of the MRs have to be identified. Other companies can implement the process by defining specific states and keeping track of them. In the case of the studied company we were able to apply the measurements out of the box based on the tracking system already existing. In future work lean software maintenance needs to be investigated in different industrial contexts.

9.8 References

[1] Reiner R. Dumke, Alain April, and Alain Abran. Software maintenance productivity measurement: how to assess the readiness of your organization. In *Proceedings of the International Conference on Software Process and Product Measurement (IWSM/Metrikon 2004)*, pages 1–12, 2004.

[2] Omar Alam, Bram Adams, and Ahmed E. Hassan. Measuring the progress of projects using the time dependence of code changes. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2009)*, pages 329–338, 2009.

[3] David J. Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results (The Coad Series)*. Prentice Hall PTR, 2003.

[4] Victor R. Basili. The experience factory and its relationship to other quality approaches. *Advances in Computers*, 41:65–82, 1995.

[5] Taizan Chan. Beyond productivity in software maintenance: Factors affecting lead time in servicing users' requests. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2000)*, pages 228–235, 2000.

[6] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006.

[7] Joel Henry, Robert Blasewitz, and David Kettinger. Defining and implementing a measurement-based software process. *Software Maintenance: Research and Practice*, 8:79–100, 1996.

[8] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Pearson Education, 2003.

[9] James M Morgan and Jeffrey K. Liker. *The Toyota product development system: integrating people, process, and technology*. Productivity Press, New York, 2006.

[10] Shahid Mujtaba, Robert Feldt, and Kai Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *Proceedings of the Australian Software Engineering Conference (ASWEC 2010) (accepted)*, 2010.

[11] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, pages 401–404, 2009.

[12] Kai Petersen and Claes Wohlin. Software process improvement through the lean measurement (SPI-LEAM) method. *Journal of Systems and Software, in print*, 2010.

[13] Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, Boston, 2003.

[14] Mary Poppendieck and Tom Poppendieck. *Implementing lean software development: from concept to cash*. Addison-Wesley, 2007.

[15] Mary Poppendieck and Tom Poppendieck. *Leading lean software development: results are not the point*. Addison-Wesley, Upper Saddle River, NJ, 2010.

[16] Hans D. Rombach and Bradford T. Ulery. Improving software maintenance through measurement. *Proceedings of the IEEE*, 77(4):581 – 95, 1989.

[17] Norman F. Schneidewind. Measuring and evaluating maintenance process using reliability, risk, and test metrics. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 1997)*, pages 232–242, 1997.

[18] T. Scott and D. Farley. Slashing software maintenance costs. *Business Software Review*, 1988.

[19] Harry M. Sneed. Measuring the performance of a software maintenance department. In *Proceedings of the First Euromicro Conference on Software Maintenance and Reengineering (EUROMICRO 1997)*, pages 119 –127, 1997.

[20] George E. Stark. Measurements for managing software maintenance. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 1996)*, pages 152–162, 1996.

[21] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslen. *Experimentation in Software Engineering: An Introduction (International Series in Software Engineering)*. Springer, 2000.

[22] Robert K. Yin. *Case Study Research: Design and Methods, 3rd Edition, Applied Social Research Methods Series, Vol. 5*. Prentice Hall, 2002.

Appendix A

Interview Protocol

A.1 Introduction

- Explain the nature of the study to the respondent, telling how or through whom he came to be selected:
  - *Goal of the study:* Understanding hindering factors in the different development models (traditional, streamline, streamline enhanced).
  - *What is done:* Compare the different models against each other in terms of bottlenecks, avoidable rework and unnecessary work.
  - *Benefit for the interviewee:* Interview is the basis for further improving the different models considering the different views of people within the organization, gives interviewee the chance to contribute to the improvement of the model they are supposed to apply in the future
- Give assurance that respondent will remain anonymous in any written reports growing out of the study, and that his responses will be treated with strictest confidence.
- Indicate that he may find some of the questions far-fetched, silly or difficult to answer, for the reason that questions that are appropriate for one person are not always appropriate for another. Since there are no right or wrong answers, he is not to worry about these but to do as best he can with them. We are only interested in his opinions and personal experiences.

- Interviewee is to feel perfectly free to interrupt, ask clarification of the interviewer, criticize a line of questioning etc.
- Interviewer is to ask permission to tape record the interview, explaining why he wishes to do this.

A.2 Warm-up and Experience

- What is your professional background (how long at the company, education)?
- What is your role within the development life-cycle at Ericsson (short description)? Include information such as department, discipline (there are a number of pre-defined disciplines at the company for different development activities). How long have you been working in this role?
- In which other disciplines have you been working and for how long?
- What is your experience with traditional development and streamline development? Select from the following options with multiple selections being possible (has to be done once for each model):
  - No previous experience
  - Studied documentation
  - Informal discussion with colleagues
  - Seminar and group discussions
  - Used in one project (started or completed)
  - Used in several projects

A.3 Main Body of the Interview

A.3.1 Plan-Driven Development

The first question concerns bottlenecks.

Definition provided to the interviewee: Bottlenecks is a phenomena that hinders the performance or capacity of the entire development lifecycle due to a single component causing it (=bottleneck). Bottlenecks are therefore a cause for reduction in throughput.

Question: What are three critical bottlenecks you experienced / you think are present in the traditional way of working (plan-driven)?

When describing three bottlenecks, please focus on:

- Which product was developed?
- Where in the development process does the bottleneck occur?
- Why is it a bottleneck (ask for the cause)?
- How does the bottleneck affect the overall development lifecycle?

The following questions concern waste. When talking about waste, we distinguish two types of waste we would like to investigate. These types of waste are unnecessary work and avoidable rework. A definition for each type is presented to the interviewee.

Type 1 - Avoidable Rework: *Investigating avoidable rework helps us to answer: "are we doing things right"? That is, if something has been done incorrectly, incompletely or inconsistently then it needs to be corrected through reworked.*

Question: What avoidable rework (three for each) has been done in the plan-driven approach?

When describing the avoidable rework, please focus on:

- Which product was developed?
- Where in the development process is the avoidable rework done?
- What was done incorrectly, incompletely or inconsistently?
- Why is the rework avoidable?

Type 2 - Unnecessary work: *Investigating unnecessary work helps us to answer: "are we doing the right things"? That is, unnecessary work has been conducted that does not contribute to customer value. It is not avoidable rework, as it is not connected to correcting things that have been done wrong.*

Question: What is unnecessary work (three for each) done in the plan-driven approach?

When describing the unnecessary work, please focus on:

- Which product was developed?
- Where in the development process is the unnecessary work done?
- Why is the unnecessary work executed?
- How is the unnecessary work used in the development?

A.3.2 Incremental and Agile Approach

After having identified the critical issues in plan-driven development we would like to capture what the situation is after introducing the new incremental and agile practices.

Note: In this part the same questions were asked as was the case for the plan-driven approach, now focusing on the situation after migrating to the incremental and agile approach.

A.4 Closing

Is there anything else you would like to add that you think is interesting in this context, but not covered by the questions asked?

Appendix B

Qualitative Analysis

Figure B.1 illustrates the analysis steps presented in the description of the research methodology with an example of the identification of factor F01 shown in Table 4.6.

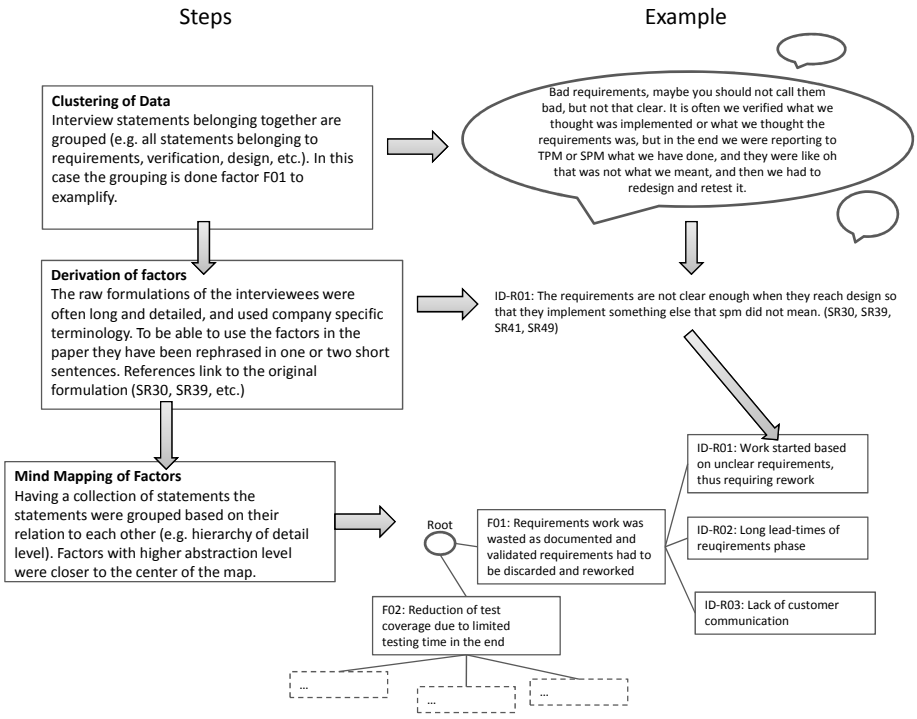


Figure B.1: Example of Analysis Illustrating the Identification of Factor F01

List of Figures

List of Tables

Without the right information, you're just another person with an opinion.

–Tracy O'Rourke, CEO of Allen-Bradley

ABSTRACT

Background: The software market is becoming more dynamic which can be seen in frequently changing customer needs. Hence, software companies need to be able to quickly respond to these changes. For software companies this means that they have to become agile with the objective of developing features with very short lead-time and of high quality. A consequence of this challenge is the appearance of agile and lean software development. Practices and principles of agile software development aim at increasing flexibility with regard to changing requirements. Lean software development aims at systematically identifying waste to focus all resources on value adding activities.

Objective: The objective of the thesis is to evaluate the usefulness of agile practices in a large-scale industrial setting. In particular, with regard to agile the goal is to understand the effect of migrating from a plan-driven to an agile development approach. A positive effect would underline the usefulness of agile practices. With regard to lean software development the goal is to propose novel solutions inspired by lean manufacturing and product development, and to evaluate their usefulness in further improving agile development.

Method: The primary research method used throughout the thesis is case study. As secondary methods for data collection a variety of approaches have been used, such as semi-structured interviews, workshops, study of process documentation, and use of quantitative data.

Results: The agile situation was investigated through a series of case studies. The baseline situation (plan-driven development) was evaluated and the effect of the introduction of agile practices was captured, followed by an in-depth analysis of the new situation. Finally, a novel approach, Software Process Improvement through the Lean Measurement (SPI-LEAM) method, was introduced providing a comprehensive measurement approach supporting the company to manage their work in process and capacity. SPI-LEAM focuses on the overall process integrating different dimensions (requirements, maintenance, testing, etc.). When undesired behavior is observed a drill-down analysis for the individual dimensions should be possible. Therefore, we provided solutions for the main product development flow and for software maintenance. The lean solutions were evaluated through case studies.

Conclusion: With regard to agile we found that the migration from plan-driven to agile development is beneficial. Due to the scaling of agile new challenges arise with the introduction. The lean practices introduced in this thesis were perceived as useful by the practitioners. The practitioners identified concrete decisions in which the lean solutions could support them. In addition, the lean solutions allowed identifying concrete improvement proposals to achieve a lean software process.

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

ISSN 1653-2090

2010:04

ISBN 978-91-7295-180-8