The Journal of Systems and Software 82 (2009) 1479–1490

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case

Kai Petersen [a,b,*], Claes Wohlin [a]

[a] School of Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden
[b] Ericsson AB, Box 518, SE-371 23, Sweden

ARTICLE INFO

Article history:
Received 17 November 2008
Received in revised form 6 February 2009
Accepted 23 March 2009
Available online 1 April 2009

Keywords:
Agile
Incremental
State of the art
Case study

ABSTRACT

Recent empirical studies have been conducted identifying a number of issues and advantages of incremental and agile methods. However, the majority of studies focused on one model (Extreme Programming) and small projects. To draw more general conclusions we conduct a case study in large-scale development identifying issues and advantages, and compare the results with previous empirical studies on the topic. The principle results are that (1) the case study and literature agree on the benefits while new issues arise when using agile in large-scale and (2) an empirical research framework is needed to make agile studies comparable.

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 (Dybå and Dingsøyr, 2008) 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 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 division of internal and external releases, small and motivated teams developing software in three-month projects (time boxing), 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 paper is shown in Fig. 1: Section 2 presents the state of the art, summarizing issues and advantages identified in literature. Thereafter, the investigated process model is

* Corresponding author. Address: School of Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden. Tel.: +46 10 7140572.
E-mail addresses: kai.petersen@bth.se, kai.petersen@ericsson.com (K. Petersen), claes.wohlin@bth.se (C. Wohlin).
URLs: http://www.bth.se/besq, http://www.ericsson.com (K. Petersen), http://www.bth.se/besq (C. Wohlin).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.03.036

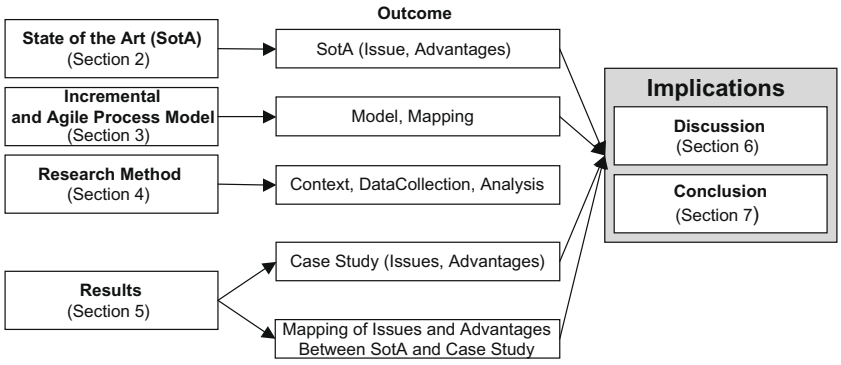

Fig. 1. Structure of the paper.

described in Section 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 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) 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 2 (see Fig. 1). Sections 6 and 7 present the implications of the results. The implications of the comparison of state of the art and case are discussed, and the mapping (Section 3) can be used to discuss generalizability and comparability of results.

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 (Dybå and Dingsøyr, 2008). 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 (Pikkarainen et al., 2008). 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 1 provides an overview of advantages of agile and incremental development that have been empirically shown in the studies identified in Dybå and Dingsøyr (2008). The advantages that have been shown for agile methods are clearly dominated by studies that investigated XP or a modified version of XP (Karlström and Runeson, 2005). 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 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 1 and 2 are used as an input for the comparison with the results from the case study in Section 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.

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

Table 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	Bahli and Abou-Zeid (2005), Karlström and Runeson (2005), Svensson and Höst (2005) and Pikkarainen et al. (2008)
A02	Customers are perceived by programmers as very valuable allowing developers to have discussions and get early feedback	XP/SC/XP/XP	–/6/–/6	Karlström and Runeson, 2005, Mann and Maurer (2005), Svensson and Höst (2005) and Tessem (2003)
A03	Pair programming facilitates learning if partners are exchanged regularly	XP	6	Tessem (2003)
A04	Process control, transparency, and quality are increased through continuous integration and small manageable tasks	XP	–	Karlström and Runeson (2005)
A05	XP is very much technical-driven empowering engineers and thus increases their motivation	XP	–	Karlström and Runeson (2005)
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)	–	Svensson and Höst (2005)
A08	The social job environment is perceived as peaceful, trustful, responsible, and preserving quality of working life	XP	23	Robinson and Sharp (2004)
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	Ilieva et al. (2004)
A10	Developers perceive the job environment as comfortable and they feel like working more productive using pair programming	XP	–	Mannaro et al. (2004)
A11	Student programmers perceive the quality of code higher using pair programming	XP	–	Mannaro et al. (2004)

Table 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)	–	Svensson and Höst (2005)
I02	Architectural design does not have enough focus in agile development leading to bad design decisions	Gen./Gen.	–/–	McBreen (2003) and Stephens and Rosenberg (2003)
I03	Agile development does not scale well	Gen.	–	Cohen et al. (2004)
I04	Pair programming is perceived as exhaustive and inefficient	XP/XP/XP	4/12/6	Ilieva et al. (2004), MacKenzie and Monk (2004) and Tessem (2003)
I05	Team members need to be highly qualified to succeed using agile	XP	6	Merisalo-Rantanen et al. (2005)
I06	Teams are highly coherent which means that the communication within the team works well, but inter-team communication suffers	XP/SC&XP	–/7	Karlström and Runeson (2005) and Pikkarainen et al. (2008)
I07	The empowerment of engineers makes managers afraid initially, and thus requires sufficient training of managers	XP	–	Karlström and Runeson (2005)
I08	Implementation starts very early, thus technical issues are raised too early from a management point of view	XP	–	Karlström and Runeson (2005)
I09	On-site customers have to commit for the whole development process which puts them under stress	XP	16	Martin et al. (2004)
I10	From the perspective of students, pair programming is not applicable if one partner is much more experienced than the other	XP	–	Melnik and Maurer (2002)

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 Fig. 2 an overview of development process is provided. At the top of Fig. 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,

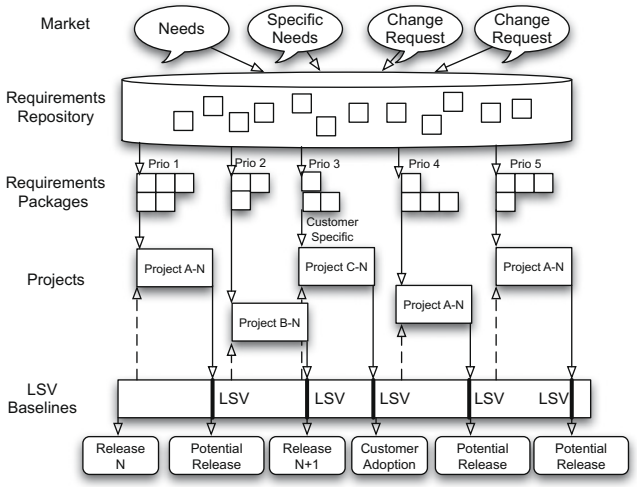

Fig. 2. Development process.

Table 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		√		

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.

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 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 Larman (2003)) 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.

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 predefined 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 6.

4. Research method

The research method used was case study. The design of the study follows the guidelines provided for case study research in Yin (2002).

4.1. Case study context

As a complement to the process model description, the context of the study was 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.

4.2. Research questions and propositions

This study aimed 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 were 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 was 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.

4.3. Case selection and units of analysis

Three subsystems that are part of a large-scale product were studied at the company. The large-scale product was the case being studied while the three subsystems were distinct units of analysis embedded in the case. Table 4 summarizes some characteristics of the case and units of analysis. The LOC measure only included 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 showed that the case and its units of analysis can be considered large-scale, the overall system being 20 times larger than Apache.

Table 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

4.4. Data collection procedures

The data was collected through interviews and from process documentation.

4.4.1. 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 were randomly selected from the list. The more persons were available for one role the more persons were selected. The reason for doing so was to not disturb the projects, that is if only one person was 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 were 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. The roles were 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.

4.4.2. Interview design

The interview consisted of five parts, the duration of the interviews was set to approximately 1 h. In the first part of the interview the interviewees were provided with an introduction to the

Table 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

purpose of the study and explanation why they were 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 were 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.

4.4.3. Process documentation

The company provides process documentation to their employees, as well as presentations on the process for training purposes. We studied this documentation to facilitate a good understanding of the process in the organization. Furthermore, presentations given at meetings were investigated, showing the progress and first results of introducing agile and incremental practices from a management perspective. However, the main source of information was the interviews, with the process documentation mainly used to get a better understanding of the process and to triangulate what was said in the interviews. The documentation and talking to people in the organization resulted in the description of the process model in Section 3.

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 (Dybå and Dingsøyr, 2008) was used as an input. As a starting point the advantages and disadvantages were extracted from the review (SotA). To identify more advantages and issues, the results and discussion sections of the identified papers in the review were 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 was 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 was 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 paper transcribed more than 30 h of audio recordings from the interviews which were 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 was clustered, grouping statements belonging together. For example, statements related to requirements engineering were grouped together. Thereafter, statements addressing similar areas were grouped. To provide an example, three statements related to requirements prioritization are shown in the textbox 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 contained detailed explanations and therefore was abstracted by deriving factors from the raw data. Each factor was 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) were 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 were 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 Fig. 3). This branch resulted in issue CI02 in Table 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 were validated in two workshops with three representatives from the company. The representatives had an in-depth knowledge of the processes. Together, the first three steps of analysis described here were

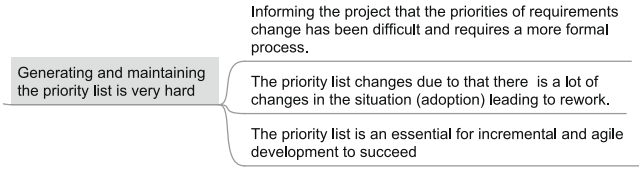

Fig. 3. Cutout from mind map.

reproduced with the authors and company representatives. As input for the reproduction of factors, a subset of randomly selected issues and advantages were 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.

Finally, the SotA and case study results were 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.

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 Yin (2002). 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 was 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 was a threat that the presence of the researcher influences the outcome of the study. The threat was 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 was also threatened if interview questions are misunderstood or misinterpreted. To mitigate the threat pre-tests of the interview were conducted.

Internal validity: Internal validity is primarily for explanatory and causal studies, where the objective is to establish a causal relationship. As this study was 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 was an adoption of practices from different general process models (see Section 3). Care was 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 paper to other development processes their context must be taken into account. Furthermore, a potential threat was that the actual case study was conducted within one company. To minimize the influence of the study being conducted at one company, the objective was 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 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 was designed so that data was collected from different sources, i.e. to conduct triangulation to ensure the correctness of the findings. The interviews were recorded and the correct interpretation of the data has been validated through workshops with representatives of the company.

5. Results

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

5.1. Advantages

Table 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

Table 6
Advantages identified in case study.

ID	Advantages	SotA (ID)
CA01	Small projects and 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

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 transferred 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).

5.2. Issues

The issues identified in this case study as well as the references to similar issues of SotA are shown in Table 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

Table 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

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).

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.

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 and Dingsøyr (2008) 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).

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. 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 (3 out of 12) 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. (2005) 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.

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 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.

7. Conclusions and future work

This paper 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.

Appendix A. Interview guide

A.1. Phase 1: introduction

Interviewer tells respondent something about himself, his background, training, and interest in the area of inquiry.

A.1.1. Presentation of study goals

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 agile and incremental development model. We seek a broad overview of what issues are there regarding hindering factors and how severe the issues are in comparison, i.e. we are not looking into each of them in detail. Therefore, we would like the interviewee to provide us with concise descriptions for each of the factors.
- Benefit for the interviewee: 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. Your view counts!

A.1.2. General information of the interview process

- Indicate that the interviewee 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, the

interviewee is not to worry about these but to do as best he or she can with them. We are only interested in his or her opinions and personal experiences.
- Interviewee is to feel perfectly free to interrupt, ask clarification of the interviewer, criticize a line of questioning, etc.
- Give assurance that the interviewee will remain anonymous in any written reports growing out of the study, and that his or her responses will be treated with strictest confidence, this is true for everybody (managers, colleagues, etc.)
- Interviewee will receive Feedback regarding the study in form of a presentation and discussion of the results obtained.

A.1.3. Taping

Provide motivation for taping: Increases the validity of the study as otherwise interpretation of what the interviewee says takes place twice (when taking notes reformulate things, and also when interpreting the notes). Goal is to put forward what the interviewee wants to say, not what we think he wants to say!

A.2. Phase 2: warm-up

Question 1: What is your professional background:

- How long at company
- Previous education

Question 2: What is your role within the development lifecycle at Ericsson (short description)?

- Department
- Discipline (there are a number of pre-defined disciplines, like requirements, design, etc., at Ericsson)
- Work activities/responsibilities

Question 3: In which areas of software development have you worked in the past (e.g. requirements engineering, design, implementation and so forth)?

Question 4: In which areas of software engineering do you consider yourself most experienced?

Question 5: What is your experience with the development process? Here the interviewee has to rate his or her experience on the following scale:

1. No previous experience
2. Studied documentation
3. Seminar and group discussions
4. Used the model in one project
5. Used the model in several projects
6. Other

A.3. Phase 3: main body of the interview

A.3.1. Questions regarding 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 1: What are three bottlenecks you experienced/you think are present in the incremental and agile development process?

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?

A.3.2. Questions regarding the production of 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 1: What avoidable rework (three for each) has been done/ will be done in the incremental and agile development process?

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 1: What is unnecessary work (three for each) done in the incremental and agile development process?

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.4. Phase 1: closing

Question: Is there anything else you would like to add that you think is interesting in this context, but not covered by the questions asked?

References

Bahli, B., Abou-Zeid, E.-S., 2005. The role of knowledge creation in adopting xp programming model: an empirical study. In: ITI Third International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society, 2005.

Cataldo, M., Wagstrom, P., Herbsleb, J.-D., Carley, K.-M., 2005. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work (CSCW 2006), pp. 353–362.

Cohen, D., Lindvall, M., Costa, P., 2004. Advances in Computers, Advances in Software Engineering. An Introduction to Agile Methods. Elsevier, Amsterdam. Chapter.

Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: a systematic review. Information and Software Technology 50 (9–10), 833–859.

Ilieva, S., Ivanov, P., Stefanova, E., 2004. Analyses of an agile methodology implementation. In: Proceedings of the 30th EUROMICRO Conference (EUROMICRO 2004), pp. 326–333.

Karlström, D., Runeson, P., 2005. Combining agile methods with stage-gate project management. IEEE Software 22 (3), 43–49.

Larman, C., 2003. Agile and Iterative Development: A Manager's Guide. Pearson Education.

MacKenzie, A., Monk, S.R., 2004. From cards to code: how extreme programming re-embodies programming as a collective practice. Computer Supported Cooperative Work 13 (1), 91–117.

Mann, C., Maurer, F., 2005. A case study on the impact of scrum on overtime and customer satisfaction. In: Proceedings of the AGILE Conference (AGILE 2005), pp. 70–79.
Mannaro, K., Melis, M., Marchesi, M., 2004. Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies. In: Proceedings of the Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005), pp. 166–174.
Martin, A., Biddle, R., Noble, J., 2004. The xp customer role in practice: three studies. In: Agile Development Conference, pp. 42–54.
McBreen, P., 2003. Questioning Extreme Programming. Pearson Education, Boston, MA, USA.
Melnik, G., Maurer, F., 2002. 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), pp. 241–250.
Merisalo-Rantanen, H., Tuunanen, T., Rossi, M., 2005. Is extreme programming just old wine in new bottles: a comparison of two cases. Journal of Database Management 16 (4), 41–61.
Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J., 2008. The impact of agile practices on communication in software development. Empirical Software Engineering 13 (3), 303–337.
Robinson, H., Sharp, H., 2004. The characteristics of xp teams. In: Proceedings of the 5th International Convergence on Extreme Programming and Agile Processes in Software Engineering (XP 2004), pp. 139–147.
Stephens, M., Rosenberg, D., 2003. Extreme Programming Refactored: The Case Against XP. Apress, Berkeley, CA.
Svensson, H., Höst, M., 2005. 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), pp. 256–264.
Tessem, B., 2003. Experiences in learning xp practices: a qualitative study. In: Proceedings of the Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004), pp. 131–137.
Yin, R.K., 2002. Case Study Research: Design and Methods, third ed. Applied Social Research Methods Series, vol. 5 Prentice Hall.

Kai Petersen is an industrial Ph.D. student at Ericsson AB and Blekinge Institute of Technology. He received his Master of Science in Software Engineering (M.Sc.) from Blekinge Institute of Technology. Thereafter, he worked as a research assistant at University of Duisburg Essen, focusing on software product-line engineering and service-oriented architecture. His current research interests are empirical software engineering, software process improvement, lean and agile development, and software measurement.

Claes Wohlin is a professor of software engineering and the Pro Vice Chancellor of Blekinge Institute of Technology, Sweden. He has previously held professor chairs at the universities in Lund and Linköping. His research interests include empirical methods in software engineering, software metrics, software quality, and requirements engineering. He received a Ph.D. in communication systems from Lund University. He is Editor-in-Chief of Information and Software Technology and member of three other journal editorial boards. He was the recipient of Telenors Nordic Research Prize in 2004 for his achievements in software engineering and improvement of reliability for telecommunication systems.