Empir Software Eng (2010) 15:654–693
DOI 10.1007/s10664-010-9136-6

The effect of moving from a plan-driven to an incremental software development approach with agile practices

An industrial case study

Kai Petersen · Claes Wohlin

Published online: 10 July 2010

Editor: Forrest Shull

Abstract So far, only few in-depth studies focused on the direct comparison of process models in general, and between plan-driven and incremental/agile approaches in particular. That is, it is not made explicit what the effect is of moving from one model to another model. Furthermore, there is limited evidence on advantages and issues encountered in agile software development, this is specifically true in the context of large-scale development. The objective of the paper is to investigate how the perception of bottlenecks, unnecessary work, and rework (from hereon referred to as issues) changes when migrating from a plan-driven to an incremental software development approach with agile practices (flexible product backlog, face-to-face interaction, and frequent integration), and how commonly perceived these practices are across different systems and development roles. The context in which the objective should be achieved is large-scale development with a market-driven focus. The selection of the context was based on the observation in related work that mostly small software development projects were investigated and that the investigation was focused on one agile model (eXtreme programming). A case study was conducted at a development site of Ericsson AB, located in Sweden in the end of 2007. In total 33 interviews were conducted in order to investigate the perceived change when migrating from plan-driven to incremental and agile software development, the interviews being the primary source of evidence. For triangulation purposes measurements collected by Ericsson were considered, the measurements relating to unnecessary work (amount of discarded requirements) and rework (data on testing efficiency and maintenance effort). Triangulation in this context means that the measurements were used to confirm the perceived changes with an additional data source. In total 64 issues were identified, 24 being of general nature and the remaining 40 being local and therefore unique to individual's opinions or a specific

K. Petersen (✉) · C. Wohlin
Blekinge Institute of Technology, 372 25 Ronneby, Sweden
e-mail: kai.petersen@bth.se

C. Wohlin
e-mail: claes.wohlin@bth.se

system. The most common ones were documented and analyzed in detail. The commonality refers to how many persons in different roles and across the systems studied have mentioned the issues for each of the process models. The majority of the most common issues relates to plan-driven development. We also identified common issues remaining for agile after the migration, which were related to testing lead-time, test coverage, software release, and coordination overhead. Improvements were identified as many issues commonly raised for the plan-driven approach were not raised anymore for the incremental and agile approach. It is concluded that the recent introduction (start in 2005 with the study being conducted in the end of 2007) of incremental and agile practices brings added values in comparison to the plan-driven approach, which is evident from the absence of critical issues that are encountered in plan-driven development.

Keywords Incremental · Agile · Plan-driven · Case study · Migration

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 Svensson and Höst 2005; Karlström and Runeson 2005, and Benediktsson et al. 2006).

Due to the increased importance and interest in agility of software development a systematic review (Dybå and Dingsøyr 2008) 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 an approach employing incremental and agile practices. 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 Ericsson for several years. Due to industry benchmarks and thereby identified performance issues (e.g. related to lead-times) Ericsson first adopted incremental practices starting in the middle of 2005. Agile practices (flexible product-backlog, face-to-face interaction, and frequent integration) were added in late 2006 and early 2007. Overall, we will show that Ericsson's model shares 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 Ericsson 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 Ericsson 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 provided in Yin (2002). Furthermore, we used the guidelines provided specifically in a software engineering context by Runeson and Höst (2009).

The contributions of the paper and case study are:

- Illustrate an industrial approach of using incremental and agile practices and a comparison of the industrial 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 process used after introducing incremental and agile practices at Ericsson. 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 2 presents related work. Thereafter, Section 3 illustrates the development processes used at Ericsson and compares them to known models from literature. Section 4 describes the research design. The analysis of the data is divided into one qualitative (Section 5) and one quantitative (Section 6) part. Based on the analysis, the results are discussed in Section 7. Section 8 concludes the paper.

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.

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. Hirsch 2005): 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. Thomas 2001; Jarzombek 1999; Johnson 2002). Consequences have been that the customers' current needs are not addressed by the end of the project (Jarzombek 1999), resulting in that many of the features implemented are not used (Johnson 2002). Additionally, there is a problem in integrating the overall system in the end and testing it (Jones 1995). 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 (Cohen et al. 2001).

RUP The RUP process was investigated in a case study mainly based on interviews (Hanssen et al. 2005). 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. Heijstek and Chaudron (2008) 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 2.3). Plan-driven approaches are still relevant today as they are widely used

in practice as recognized in many research articles (c.f. Raccoon 1997; Beck 1999; Laplante and Neill 2004; Dai and Guo 2007)). The case company of this study used the approach till 2005, and there are still new research publications on plan-driven approaches (cf. Hanssen et al. 2005; Heijstek and Chaudron 2008; Petersen et al. 2009).

2.2 Incremental and Agile Development

Dybå and Dingsøyr (2008) 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. Svensson and Höst 2005; Karlström and Runeson 2005; Bahli and Abou-Zeid 2005). Furthermore, the benefit of communication helps to transfer knowledge (Bahli and Abou-Zeid 2005). Agile methods further propose to have the customer on-site. This is perceived as valuable by developers as they can get frequent feedback (Tessem 2003; Svensson and Höst 2005; Karlström and Runeson 2005), and the customers appreciate being on-site as this provides them with control over processes and projects (Ilieva et al. 2004). An additional benefit is the regular feedback on development progress provided to customers (Ilieva et al. 2004). 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 (Mannaro et al. 2004).

Negative factors Well known problems are that architecture does not have enough focus in agile development (cf. McBreen 2003; Stephens and Rosenberg 2003) and that agile development does not scale well (Cohen et al. 2004). 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 (Svensson and Höst 2005). 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 (Wils et al. 2006). On the team level team members have to be highly qualified (Merisalo-Rantanen et al. 2005). 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 (Martin et al. 2004).

Petersen and Wohlin (2009a) compared issues and advantages identified in literature with an industrial case. 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.

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 (Dagnino et al. 2004). 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 (Ceschi et al. 2005; Sillitti et al. 2005). However, a drawback of agile development is that team members are not as easily interchangeable as in waterfall-oriented development (Baskerville et al. 2003).

Studies reported significant productivity gains of 42% (Ilieva et al. 2004), 46% (Layman et al. 2004), and up to 337% (Benediktsson et al. 2006). The study reporting a productivity gain (LOC/Effort) of 337% (Benediktsson et al. 2006) was a multi-project experiment with 55 computer science students. The students applied different models (V-Model representing traditional development, incremental and evolutionary models, and XP). Regarding time consumption, the results show that XP saves time on requirements, but requires more time for verification and validation. In coding, no major time differences were discovered. The huge gain in productivity was due to more code developed. However, it is important to mention that this does not imply that the team delivered more functionality. Ilieva et al. (2004) measured productivity for each iteration and compared the productivity of a baseline project (characterized as heavyweight and documentation driven) with an XP project. The productivity gains are the highest for the first two of three iterations. However, the last iteration did not lead to gains as only bug fixing and modifications were requested in this iteration. This required considerable effort, but no new code was developed. Layman et al. (2004) compared two releases with each other, one developed with traditional methods and one using XP. The results show improvement in programmer productivity by 46%. However, as pointed out in the study the increase on productivity can also be influenced by the gained experience during the development of the first release. Even though the study in this paper does not focuses on productivity, the observations in the productivity studies showed a significant positive effect of an introduction of agile development, and hence are reported here. With regard to quality (Layman et al. 2004) reported positive effects of the introduction with regard to reduction in the number of defects discovered. The number of defects pre-release (i.e. discovered defects through verification activities conducted by the development organization) was reduced by 65% and the number of defects post-release (i.e. discovered defects by the customer) was reduced by 35%.

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 (Dybå and Dingsøyr 2008). 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.

3 The Plan-Driven and Agile Models at Ericsson

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

3.1 Plan-Driven Approach

The plan-driven model that was used at Ericsson implemented the main characteristics of plan-driven approaches as summarized by Hirsch (2005). 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 occurred 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 (Tomaszewski 2006) 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 Ericsson 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 Ericsson (i.e. the baseline situation) can be found in Petersen et al. (2009).

3.2 Development Approach Using Incremental and Agile Practices

The used at Ericsson after the migration is shown in Fig. 1. The process relied on a set of company specific practices that have been introduced. The numbers (1 to 5) in Fig. 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

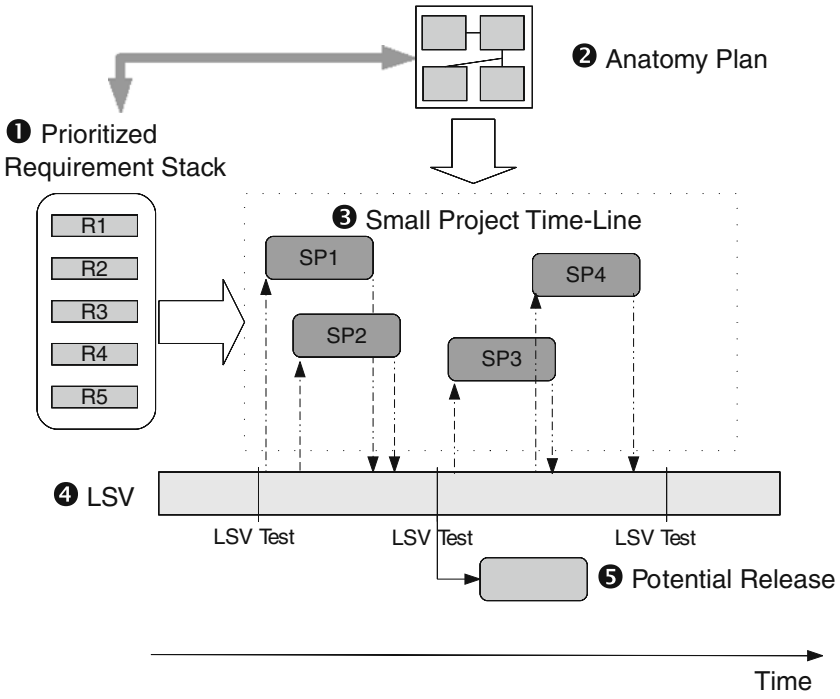

Fig. 1 Development process

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 Fig. 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.

The transition from the plan-driven to an incremental approach with additional agile practices has been done stepwise. The implementation of the incremental process formed the basis for the introduction of additional agile practices. 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 Ericsson 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.

3.3 Comparison with General Process Models

Ericsson'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 Ericsson (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 1 (created based on the information provided in Larman (2003) who provides a description of the general models) shows that 4 out of 4 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.

Ericsson'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 Ericsson'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 1 Comparison with general process models (practices identified using Larman 2003)

Principle	ID	XP	SC	C
Iterations and increments	√	√	√	√
Internal and external releases	√			√
Time boxing	√	√	√	√
No change of started projects	√		√	√
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.
- *Flexible 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 all principles with ID and realizes approximately half of the 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 (Schwaber 2004). 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 (Schwaber 2004; Larman 2003; Koch 2005). 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 (Larman 2003). 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 (Petersen et al. 2009). 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 Case Study Design

4.1 Study Context

It is of importance to describe the context in order to aid in the generalizability of the study results (cf. Petersen and Wohlin 2009b). 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

Table 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 additional agile practices
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 and detailed requirements for development teams (in both development approaches).
Requirements tracking	Requirements proprietary tool for managing requirements on product level (i.e. across projects). Requirements database is 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 1)
Incremental and agile maturity	Stepwise implementation of incremental and agile practices 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.

solutions. The company is ISO 9001:2000 certified. The development of Ericsson 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 2.

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*

practices? 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 a plan-driven to an incremental process with agile practices 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 Ericsson, 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 (Yin 2002). 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 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 Ericsson 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 introduction of the new practices.

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.

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).

Table 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

A subsystem was a large system component of the overall system. Table 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 Ericsson (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.

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 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 Ericsson. This section explains the data collection procedures for each source in detail.

4.4.1 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 Ericsson, and the experience with the old process model (plan-driven) and the new process model with the use of incremental and agile practices. 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 Ericsson. Ten persons had at least 10 years of experience working at Ericsson. 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. The roles are divided into *Requirements*, *Project Management*, *Implementation*, *Quality Assurance*, and *Life Cycle Management*.

- *Requirements:* 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.
- *Project Planning:* 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.
- *Implementation:* 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.

Table 4 Distribution of interviewees between primary responsibilities and units of analysis

	S1	S2	S3	Total
Requirements	2	1	1	4
Project planning	3	2	1	6
Implementation	3	2	1	6
Quality assurance	4	3	–	7
Life cycle management	6	4	–	10
Total	18	12	3	33

4.4.2 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 Ericsson 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 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 approach with incremental and agile practices.

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.

4.4.3 Process Documentation

Ericsson 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 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 Ericsson to the decision of migrating. 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

Table 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 (Anderson 2003).
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 (Anderson 2003; Poppendieck and Poppendieck 2003).
Avoidable rework	Rework can be avoided when doing things completely, consistently and correctly (Fairley and Willshire 2005). For example, having the right test strategy to discover faults as early as possible (Damm et al. 2006; Damm and Lundberg 2007).

Ericsson, which was also well supported by documentation; (2) To extract context information relevant for this study.

4.4.4 Performance Measures

Ericsson 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 introducing the incremental and agile practices. 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 Ericsson 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 Damm et al. (2006) 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 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.

4.5 Data Analysis

The data analysis was done in six different steps, as shown in Fig. 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

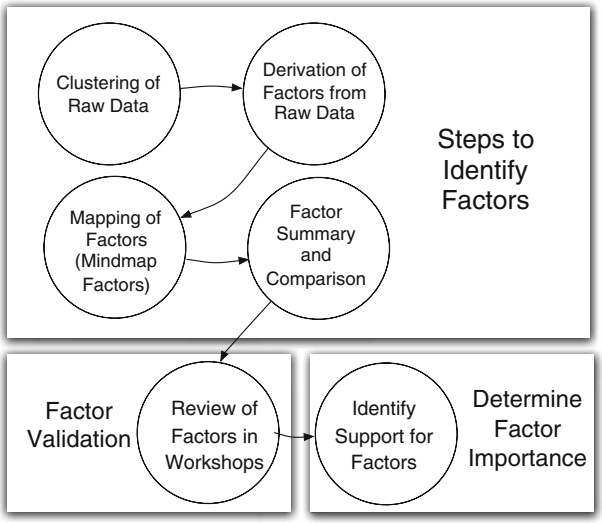

Fig. 2 Data analysis process for qualitative data

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 5 and 6).

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

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 Ericsson participated in a workshop to review the issues. All representatives from Ericsson had an in-depth knowledge

of both process models. The validation was done by randomly selecting issues and each of the representatives of Ericsson 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 identified an issue as critical for plan-driven, but not for the approach using incremental and agile practices, 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 introducing incremental and agile practices. 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 the use of incremental and agile practices. 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.

4.6 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 (Yin 2002), and threats in a software engineering context is discussed in for example (Wohlin et al. 2000). 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.

4.6.1 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 Ericsson 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 Ericsson 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.

4.6.2 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. 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.

4.6.3 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 new approach using incremental and agile practices were adaptations of general processes presented in the literature. This is obvious when it comes to the incremental and agile practices, 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 approach 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 1 illustrates which practices from the general process models have been employed at Ericsson. 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 Ericsson 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.

4.6.4 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 4.5). 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.

4.6.5 Summary

In summary, the case study has been designed according to guidelines and tactics provided in Yin (2002). 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 Ericsson, 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.

5 Qualitative Data Analysis

Section 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 6. In the column "Classification" we stated the number of interviewees and the number of systems for each issue in the brackets.

Commonly perceived improvements are shown in Table 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 anymore when using the incremental and agile practices introduced at Ericsson. 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, the introduction of the new practices enabled early testing and

Table 6 Classification of identified issues

ID	Classification	Model	Process area	Description
F01	General (12/3)	PD	Requirements	Requirements work was wasted as documented and validated requirements had to be discarded or reworked.
F02	General (13/2)	PD	Verification	Reduction of test coverage due to limited testing time in the end.
F03	Very common (10/2)	PD	Verification	Amount of faults found increased with late testing.
F04	Very common (7/3)	PD	Verification	Faults found late in the process were hard and expensive to fix.
F05	Very common (7/2)	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 (6/3)	PD	Requirements	Too much documentation was produced in requirements engineering that was not used in later stages of the process.
F07	Common (6/3)	PD	Design	Design had free capacity due to long requirements engineering lead times.
F08	Common (4/3)	PD	Design	Confusion on who implemented which version of the requirements.
F09	Common (4/2)	PD	Maintenance	High number of corrections for faults reported by customers were released.
F10	Common (4/2)	PD	Project mgt.	Specialized competence focus and lack of confidence.
F11	Common (4/3)	IA	Verification	Low test coverage.
F12	Common (4/2)	IA	Release	Release was involved too late in the development process.
F13	Common (4/2)	IA	Project mgt.	Management overhead due to a high number of teams requiring much coordination and communication.

Table 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.

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.

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 (plan-driven) 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."*

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.

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 the new approach using incremental and agile practices:

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.

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 thatŠs 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 the use of incremental and agile practices are:

F11 *Low test coverage:* The reasons for low test coverage changed with the introduction of the new practices 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 migration.

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 the new development approach. 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.

5.4 Comparison of Issues

Table 6 clearly shows that a majority of general problems was related to plan-driven development. Furthermore, only one issue raised for the situation with incremental and agile practices 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 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 7.

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 use of incremental and agile practices 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.”*

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 7.

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

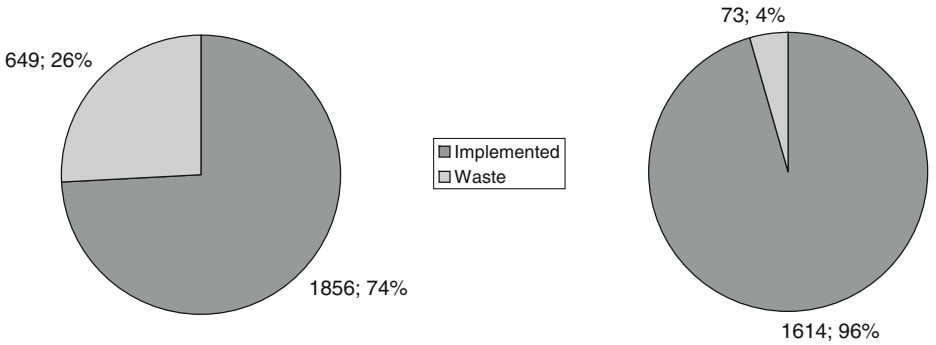

Fig. 3 Requirements waste–plan-driven (*left*) vs. incremental and agile practices (*right*)

of requirements that were implemented and discarded are shown in Fig. 3. The data includes two products as well as two generations of an additional product developed at the studied development site.

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.

6.2 Software Quality

Table 8 shows the fault-slip-through before and after the introduction of agile and incremental practices. 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.

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 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

Table 8 Fault slip before system test / LSV

Test	Number of faults	Slippage (%)
System test (plan-driven)	30	31
LSV (incremental and agile)	20	19

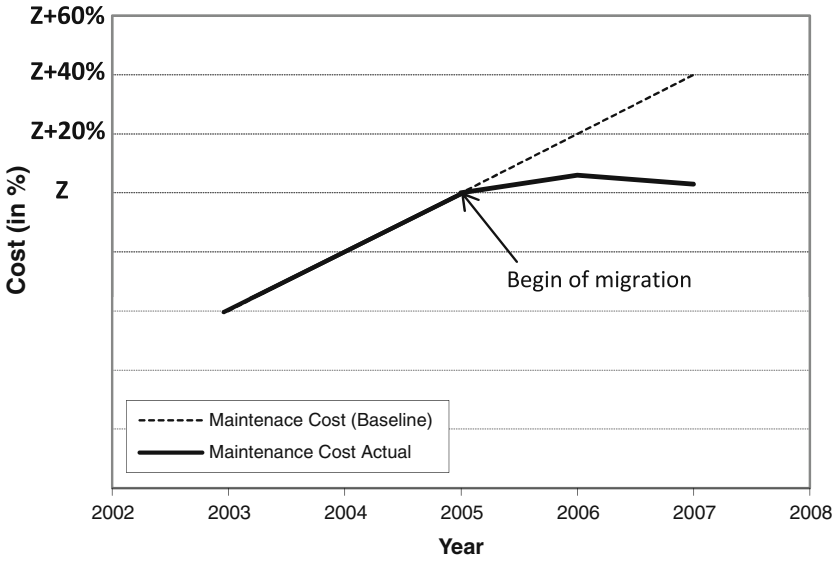

Fig. 4 Maintenance effort

years into the future (dashed line showing the maintenance cost baseline) the costs would be 40% higher than in the year 2005.

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.

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 the migration.

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, Ericsson 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 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 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 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 new development approach 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 Fig. 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 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 when using incremental and agile practices, 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 Merisalo-Rantanen et al. 2005). 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 (Svensson and Höst 2005).

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.

7.2 Open Issues

Based on the classification, the most important issues that remained after the migration were 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. After introducing incremental and agile practices 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 the new development approach. However, many projects working toward the same goal have to be coordinated. As discussed earlier (see F13 in Section 5) this requires much communication and planning involving many different people. Therefore, this issue is specifically related to the scalability of incremental and agile approaches. 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 Ericsson 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.

7.3 Implications

The results of the case study indicate that it is beneficial for large-scale organization using a plan-driven approach to introduce incremental and agile practices. 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 introducing incremental and agile practices. 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.

8 Conclusions and Future Work

This paper investigates the effect of introducing incremental and agile practices in an organization that has been working in a plan-driven way. The study shows that the most commonly perceived problems in the development models can be found in plan-driven development, and introducing incremental and agile practices 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 approach using incremental and agile practices. 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 approaches.

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 situation after introducing incremental and agile practices.

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 facilitates better understanding and allows to reduce documentation. However, the use of incremental and agile practices 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.

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: Example of the Qualitative Analysis

Figure 5 illustrates the analysis steps presented in the description of the research methodology with an example of the identification of factor F01 shown in Table 6.

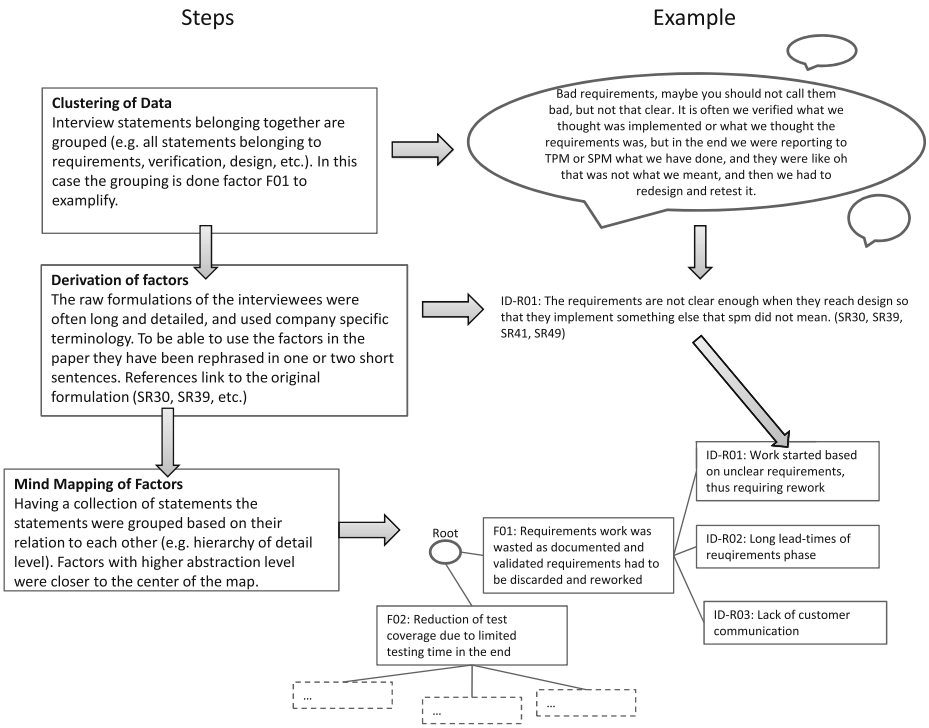

Fig. 5 Example of analysis illustrating the identification of factor F01

References

Anderson DJ (2003) Agile management for software engineering: applying the theory of constraints for business results (the coad series). Prentice Hall PTR, Englewood Cliffs

Bahli B, Abou-Zeid ES (2005) 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

Baskerville R, Ramesh B, Levine L, Pries-Heje J, Slaughter S (2003) Is internet-speed software development different? IEEE Softw 20(6):70–77

Beck K (1999) Embracing change with extreme programming. IEEE Comput 32(10):70–77

Benediktsson O, Dalcher D, Thorbergsson H (2006) Comparison of software development life cycles: a multiproject experiment. IEE Proc Softw 153(3):323–332

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

Cohen D, Larson G, Ware B (2001) Improving software investments through requirements validation. In: Proceedings of the 26th annual NASA Goddard software engineering workshop (SEW 2001). IEEE Computer Society, Washington, p 106

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

Dagnino A, Smiley K, Srikanth H, Antón AI, Williams LA (2004) 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), pp 501–506

Dai L, Guo W (2007) 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), pp 81–91

Damm LO, Lundberg L (2007) Company-wide implementation of metrics for early software fault detection. In: Proceedings of the 9th international conference on software engineering (ICSE 2007), pp 560–570

Damm LO, Lundberg L, Wohlin C (2006) Faults-slip-through—a concept for measuring the efficiency of the test process. Softw Process Improv Pract 11(1):47–59

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: a systematic review. Inf Softw Technol 50(9–10):833–859

Fairley RE, Willshire MJ (2005) Iterative rework: the good, the bad, and the ugly. IEEE Comput 38(9):34–41

Hanssen GK, Westerheim H, Bjørnson FO (2005) Using rational unified process in an sme—a case study. In: Proceedings of the 12th European conference on software process improvement (EuroSPI 2005), pp 142–150

Heijstek W, Chaudron MRV (2008) Evaluating rup software development processes through visualization of effort distribution. In: Proceedings of the 34th conference on software engineering and advanced applications (SEAA 2008), pp 266–273

Hirsch M (2005) Moving from a plan driven culture to agile development. In: Proceedings of the 27th international conference on software engineering (ICSE 2005), p 38

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

Jarzombek J (1999) The 5th annual jaws s3 proceedings

Johnson J (2002) 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)

Jones C (1995) Patterns of software systems: failure and success. International Thomson Computer Press, Boston

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

Koch AS (2005) Agile software development: evaluating the methods for your organization. Artech House, Boston

Laplante PA, Neill CJ (2004) Opinion: the demise of the waterfall model is imminent. ACM Queue 1(10):10–15

Larman C (2003) Agile and iterative development: a manager's guide. Pearson Education, Boston

Layman L, Williams LA, Cunningham L (2004) Exploring extreme programming in context: an industrial case study. In: Proceedings of the agile development conference (ADC 2004), pp 32–41

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 5th 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
Merisalo-Rantanen H, Tuunanen T, Rossi M (2005) Is extreme programming just old wine in new bottles: a comparison of two cases. J Database Manage 16(4):41–61
Petersen K, Wohlin C (2009a) A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. J Syst Softw 82(9):1479–1490
Petersen K, Wohlin C (2009b) Context in industrial software engineering research. In: Proceedings of the 3rd international symposium on empirical software engineering and measurement (ESEM 2009), pp 401–404
Petersen K, Wohlin C, Baca D (2009) The waterfall model in large-scale development. In: Proceedings of the 10th international conference on product focused software development and process improvement (PROFES 2009), pp 386–400
Poppendieck M, Poppendieck T (2003) Lean software development: an agile toolkit (the agile software development series). Addison-Wesley, Reading
Raccoon LBS (1997) Fifty years of progress in software engineering. SIGSOFT Softw Eng Notes 22(1):88–104. doi:10.1145/251759.251878
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164
Schwaber K (2004) Agile project management with Scrum. Microsoft Press, Redmond
Sillitti A, Ceschi M, Russo B, Succi G (2005) 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), p 17
Stephens M, Rosenberg D (2003) Extreme programming refactored: the case against XP. Apress, Berkeley
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 4th international conference on extreme programming and agile processes in software engineering (XP 2004), pp 131–137
Thomas M (2001) It projects sink or swim. In: British computer society review 2001
Tomaszewski P (2006) Software development productivity–evaluation and improvement for large industrial projects. PhD thesis, Dept. of Systems and Software Engineering, Blekinge Institute of Technology
Wils A, Baelen SV, Holvoet T, Vlaminck KD (2006) Agility in the avionics software world. In: XP, pp 123–132
Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A (2000) Experimentation in software engineering: an introduction (international series in software engineering). Springer, Heidelberg
Yin RK (2002) Case study research: design and methods, 3rd edn. In: Applied social research methods series, vol 5. Prentice Hall, Englewood Cliffs

Kai Petersen is an industrial PhD 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. Wohlin received a PhD in communication systems from Lund University. He is Editor-in-Chief of Information and Software Technology and member of three other journal editorial boards. Claes Wohlin was the recipient of Telenor's Nordic Research Prize in 2004 for his achievements in software engineering and improvement of reliability for telecommunication systems.