SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2008; **13**: 195–212
Published online 25 July 2007 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/spip.355

From Traditional to Streamline Development – Opportunities and Challenges

Research Section

Piotr Tomaszewski[1,2*,†], Patrik Berander[1] and Lars-Ola Damm[1,2]
[1] *School of Engineering, Blekinge Institute of Technology, SE-37225 Ronneby, Sweden*
[2] *Ericsson AB, Ölandsgatan 1, Box 518, SE-37123 Karlskrona, Sweden*

Traditional software development processes have shown to be inappropriate for markets where it is necessary to quickly respond to changing customer needs. To address this issue, a number of modern development processes have been developed that attempt to improve customer responsiveness. One such modern process is Streamline Development (SD), a process developed by and for Ericsson AB. This article presents an early evaluation of the suitability of SD for Ericsson. The evaluation was performed by finding positive and negative aspects of introducing SD, as well as identifying issues to address if implementing the new process. The data regarding the impact of introducing SD was collected in a series of interviews and then structured using a modification of Force Field Analysis (FFA). Copyright © 2007 John Wiley & Sons, Ltd.

KEY WORDS: streamline development; software development process; evaluation

1. INTRODUCTION

Today's competitive business environment in the software domain has resulted in a situation where it is becoming increasingly important to quickly respond to changing customer and market demands. In the last years, it has become evident that traditional software development processes do not handle this situation very well (Tran and Galka 1991, Graham 1992, Jensen 1995, Benediktsson and Dalcher 2003, Dalcher *et al*. 2005). The reason for this is that traditional processes tend to have rather long life cycles and do not deliver the actual customer value (i.e. the working system) until late in the process (MacCormack *et al*. 2003). At the same time, traditional processes are not very good at coping with changing requirements since they are especially exposed to changing market demands due to their long duration. In such situations, changing requirements are a source of significant amount of rework to adapt the system to new requirements (Sommerville 2004) and such rework is one of the most important productivity bottlenecks in large projects (Tomaszewski and Lundberg 2005 and Tomaszewski and Lundberg 2006). Because of this, traditional processes are not suitable in business environments where changing user needs are

* Correspondence to: Piotr Tomaszewski, Blekinge Institute of Technology, School of Engineering, Soft Center Ronneby, SE-37225 Ronneby, Sweden
† E-mail: piotr.tomaszewski@bth.se

frequent. Instead, organizations need to find alternative ways to develop their software.

To address the problems presented above, research and practice within the software domain have switched focus from such traditional sequential processes to phased processes that favor customer responsiveness (Sommerville 2004, Pfleeger 2006). In these emerging processes, the system (or part of the system) is usually available early in the development process. This makes it possible to meet customer needs faster and deliver value to the customers earlier. Frequent releases enable early customer feedback, which results in that the customers are getting more involved in the entire development process. This, in turn, makes it possible to detect and address changing needs much earlier, and thus improves customer responsiveness and reduces the risk of waste caused by implementing inadequate functionality.

The challenges with traditional processes and the potential in the emerging processes have been recognized by Ericsson AB, one of the major software developers of telecommunication systems in the world. At the time of this study, the product development units (PDUs) at Ericsson where the study was performed developed systems in a rather traditional style, in large projects that have long life cycles (often more than a year). At the same time, Ericsson is doing business within the telecommunication area, an area that is moving extremely fast forward. As an effect of this, Ericsson has experienced similar problems as the ones presented above. To address these problems, a new in-house developed process called streamline development (SD) has been suggested. SD was suggested after an extensive survey of agile and lean approaches (Section 3), and tailored to the specific characteristics of the company. The reason for developing an in-house approach was that for a company developing large projects in a market-driven environment with specific restrictions regarding how to for example manage releases, no agile approach can be applied off-the-shelf without company-specific adaptations.

This article presents a study performed at Ericsson that was planned as one of the activities in an early evaluation of SD applicability for replacing current development practices. The goal of this study was to provide decision makers with a deeper and more structured understanding of the possible effects of introducing SD. This was achieved by identifying benefits and drawbacks of implementing SD and investigating changes required to prepare the organization and its products for successful implementation of SD. The study was performed in two PDUs at Ericsson and the information regarding the impact of introducing SD was collected by interviewing representatives for the roles that would be affected by a change to SD. To structure the data collected in the interviews, a modified version of Force Field Analysis (FFA) (Johnson *et al.* 2005) was used, i.e. a method for determining the forces for and against an organizational change.

The paper is structured as follows: Section 2 presents the current (traditional) as well as the suggested SD processes at Ericsson. Section 3 describes related research while Section 4 describes the methods used in this study. In Section 5 the results of the study are presented. Section 6 discusses the findings and Section 7 concludes the work.

2. ERICSSON AND THEIR DEVELOPMENT PROCESSES

Ericsson is an ISO 9001 : 2000 certified company with main office in Sweden. Ericsson is one of the market leaders globally within the telecommunications domain and sells systems to a market with basically all mobile operators worldwide as potential customers. In order to give some background regarding the work practices at Ericsson Section 2.1 presents a description of the traditional development process that is currently used at Ericsson together with the major problems this process is causing. Section 2.2 presents SD and describes in which ways it is supposed to overcome the problems of the traditional development process described in Section 2.1. The process descriptions presented in the following sections are simplified since the main intention is to underline the major differences between the processes rather than describe them in detail.

2.1. Traditional Development Process

Figure 1 presents a simplified visualization of how the software is developed currently at Ericsson. In this process, products are normally developed in a number of consecutive releases. The product releases are developed in large, long–lasting (from 1–2 years), projects with a predefined scope. This

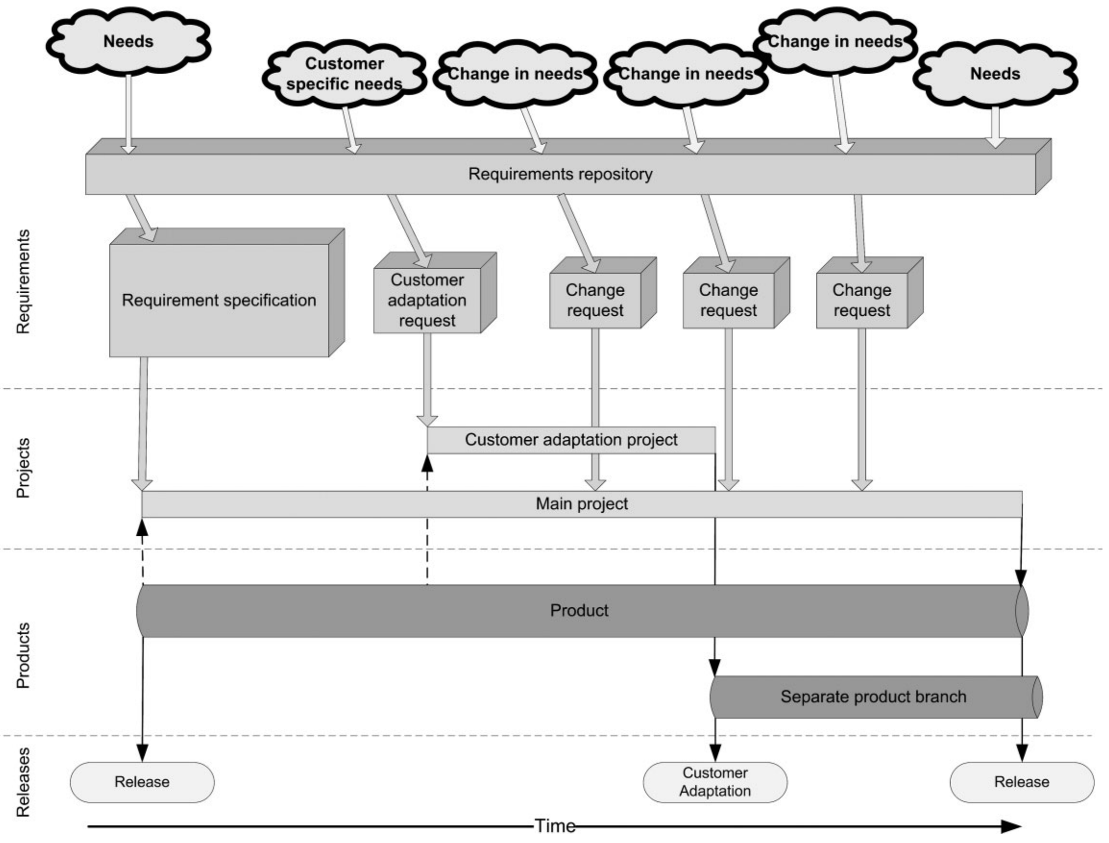

Figure 1. Traditional development process at Ericsson (one main product release) 210 × 155 mm (600 × 600 DPI)

predefined scope is typically decided by selecting a number of requirements from a requirements repository. These requirements are then refined into a requirement specification, covering an entire project. On the basis of this requirement specification, a new version of the system is developed and released to the customers.

Owing to the length of the projects and the volatile telecommunication domain, the market and its customers often change their minds while the project is still ongoing. This leads to a situation where some of the already specified and worked with requirements become obsolete. In comparison to the initial set of requirements, some requirements need to be added, some need to be changed, and some need to be deleted to better match the customer expectations. When a change in need is identified, a change request (CR) is filed (Figure 1). A CR describes an alteration from the original requirement baseline, and is conducted similarly at Ericsson as in (Wiegers 1999). When a CR enters a project, it often implies that an 'original' requirement that already has been implemented must be either reimplemented or thrown away depending on the kind of CR. In either case, already performed work is thrown away, which means that CRs entering a project lead to some kind of waste. However, it should still be recognized that it serves a good purpose since it facilitates a better product to release on the market.

When describing the overall development process above, the description primarily concerns projects aiming for a broad market launch. However, it is also common that specific customers initiate requirements that should not be included in the main release (and hence not in a main project) for some reason. Such requirements are denoted as customer adaptation requests in Figure 1 and such requirements are usually urgent and must be addressed quickly. When a decision is made to meet

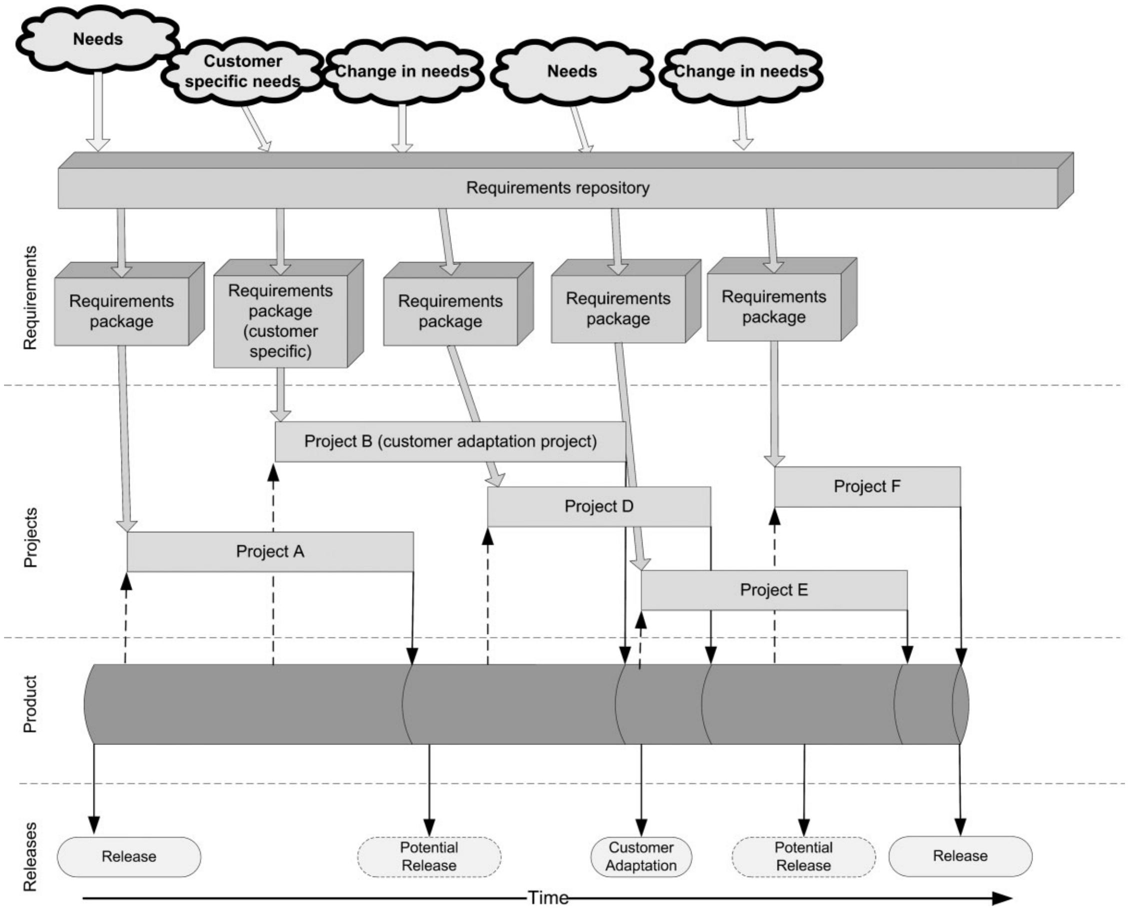

Figure 2. Streamline development process 210 × 170 mm (600 × 600 DPI)

such customer specific requirements, a customer adaptation project is started and the requested functionality is implemented into a branch of the system. Upon completion of such a project, the product is released to the ordering customer but is not integrated with the main product, and has to be maintained as a separate product branch (Figure 1).

One of the main problems Ericsson has with this traditional way of developing software is the high cost of handling CRs and specific product branches. As mentioned above, CRs identify and is the trigger of waste and rework. Owing to the long lead-times in these projects, CRs are relatively common in the current projects at Ericsson and accounts for a significant part of project cost. Furthermore, dealing with changes of already decided and sometimes implemented functionality decreases productivity and increases project lead-time. Further, even though CRs is a way of dealing with changed needs, long project lead-times decrease the company's competitiveness by lowering customer responsiveness since it is always harder to make a change of something that already is implemented than of something that is not yet specified in detail. Besides the negative impact of changes in projects, the cost of handling customer adaptations also introduces additional problems with traditional projects at Ericsson. Each such adaptation has to be maintained, almost as a separate product, which makes them very expensive.

2.2. Streamline Development Process

In SD (Figure 2), the projects are significantly shorter (at most 3 months lead-time) than in the traditional process described in Section 2.1. The size of the project will also be smaller (significantly fewer project members). This means that the scope of the projects also is reduced. A project in SD is initiated when there is a development team available and when there are a suitable number of highly prioritized requirements that can be combined into a requirements package (based on that they fit well together, etc.). The size of such a requirements package is limited by the project length – it should be possible to implement the requirements from the package within the project boundaries, i.e. in less than 3 months.

When developing according to SD, there will always be one (and only one) version of the product at any point of time. When a development project is completed, its outcome is integrated with the current product baseline (Figure 2). Such integration creates a new baseline that the next project will be integrated with. After integration it should be possible to release the system to the customers. Customer adaptation projects are handled in the same as any other project, i.e. they are integrated into the main product (see Project B in Figure 2). This means that only one latest system version needs to be maintained. However, even though each project produces a new system version that potentially can be released, it does not have to be released to the market. Therefore, in Figure 2, some of the system versions are marked as 'potential releases', which means that they can be released but do not have to be (e.g. because the cost of maintaining an additional version of a system would exceed the benefit from releasing it). Such a separation between development and release is a clear difference from the traditional development, where each project ended with a release of a new version of the system to the market. In practice, more frequent releasing can be expected in SD compared to traditional development.

SD is supposed to address a number of shortcomings of the traditional development described in Section 2.1. Most evident difference is much shorter time between identifying and implementing needs in products. In addition, as SD-projects are much shorter than traditional projects, they are less exposed to the risk of changing market demands (the change of market demands in 3 months time is considerably less probable than their change in 2 years). Further, continuous requirements prioritization of the requirements in the repository assure that only the most pressing requirements are implemented for each new release of the system, and hence each new release should be highly demanded by the customers. This increases the likelihood that requirements with lower priority that are more likely to change are not yet implemented when the market demands change. Therefore, the work on implementing them will not become wasted. SD also deals with the problem of the high cost connected with maintaining customer adaptations. In SD, customer adaptations will be integrated into the main product, which should reduce the cost connected with maintaining separate product branches.

3. RELATED WORK

One of the main drawbacks that are mentioned in relation to the traditional way of working is the long cycle time (i.e. time-to-market/customer). To address this problem, a myriad of different development approaches have been proposed, which are based on phased approaches like iterative and incremental (Pfleeger 2006). Since the late 1990s, several emerging development approaches have been focused on simplicity and flexibility, in order to develop software quickly and capably (Pfleeger 2006). Several of these approaches share the same values and beliefs, which have been documented in the 'agile manifesto' (Cockburn 2002). Some of the most known development approaches within this movement is eXtreme Programming (XP) (Beck and Andres 2005) and SCRUM (Schwaber and Beedle 2002, Schwaber 2004). Such approaches focus on delivering value early to the stakeholders. In parallel with the agile approaches, lean development has matured. Lean development stems from lean production, which was invented by Toyota (Womack *et al.* 1990). The principles of lean production from Toyota have been tailored to fit in the software engineering domain, and are presented by Poppendieck and Poppendieck (2007). In a nutshell, lean development is about reducing the development timeline by removing all nonvalue-adding wastes. Waste is anything that interferes with giving the users what they value at the time and place where it will provide most value. Hence, anything that is done that does not add value is waste, and any delay that keeps the value from the users is waste.

SD aligns well with modern (iterative and/or incremental) approaches like agile and lean. There are a lot of studies (Tran and Galka 1991, Graham 1992, Jensen 1995, Dalcher *et al.* 2005) that discuss advantages and disadvantages of such modern development practices in relation to more traditional software development processes. However, none of these studies have reported about SD, for the simple reason that SD is an internally developed process at Ericsson. Nevertheless, several studies reported describe processes that are similar to SD (e.g. processes that focus on customer responsiveness, elimination of waste). Therefore, findings and conclusions from such studies are interesting to investigate and compare with the findings from the study presented in this article.

One characteristic of SD is the ability to release more often than in traditional processes. The advantages of early and frequent releases are often discussed in the context of incremental software development. An overview of the findings concerning advantages of using incremental approaches is presented by Benediktsson and Dalcher (2003). They argue that incremental software development provides early return on investment to the customers, assures closer fit to the real customer needs, decreases the risk of rework by involving customers early in the development process, decreases the reliance on specialist personnel, lowers the dependency on external deliverables, allows better informed decision making and better understanding of trade-offs, reduces maintenance cost, and enables better resource management.

Another list of advantages of using incremental approach is presented by Graham (1992). The advantages are divided into two groups; advantages for developers and for customers. Among the benefits for developers Graham presents improved team morale, reduced maintenance, reduced risk, easier control of over-engineering, facilitated measurement of productivity, rapid feedback on the correctness of estimations, and a reduced cost of managing CRs. As advantages for customers Graham (1992) presents early availability of products, increased confidence in developer, better software quality, longer software lifetime, more flexible options, increased user acceptance, increased system assimilation, increased understanding of requirements and ability of software to meet real, not frozen needs. Apart from advantages, Graham (1992) also presents a list (largely based on Redmill 1992) of problems connected with incremental development. The problems are divided into hardware related problems, life cycle problems, management problems, and financial/contractual problems. In this article, the life cycle and management problems are of primary interest, and hence focus is put on these. Problems related to the life cycle concern problems with the requirements specification of increments, design integrity, a need for additional testing, and high cost of configuration management. Management problems, on the other hand, regard problems with: co-ordination of teams, controlling releases of increments, and prioritization and scheduling of increments as well as problems with introducing certain 'cultural changes' in the organization.

In (Middleton 2001), Middleton discusses the advantages of lean software development. Lean software development is somewhat similar to SD, as it focuses on short cycles between specifying requirements and producing software. By using lean software development, Middleton (2001) showed that it has a positive impact on quality of software. However, he also observed that introducing practices like lean software development into organizations that have traditional processes often requires changes in the organization. Such changes are required since traditional 'vertical' hierarchy may not apply to the new way of working where closer and more 'horizontal' collaboration between roles may be required. Such change may require further changes, e.g. changes in role responsibilities, promotional patterns, etc.

There are also empirical studies that attempt to quantify the gains from using modern development approaches rather than traditional ones. In (Dalcher *et al.* 2005), Dalcher *et al.* present an experiment in which a number of teams developed similar systems using different processes. In this study, the teams used a traditional approach, incremental development, evolutionary development, and extreme programming. The results showed that the use of modern approaches (i.e. incremental development, evolutionary development, and extreme programming) lowered lead-time and improved productivity. Other studies have also provided similar 'success stories' (Tran and Galka 1991, Jensen 1995, Niels 2001) in relation to implementation of modern development practices. These claim that such practices increased customer satisfaction (Tran and Galka 1991, Jensen 1995), improved product quality (Jensen 1995, Niels 2001), and decreased cost (Jensen 1995).

There are also studies reporting about situations where an organization has a working traditional process and considers moving toward a modern process. For example, Boehm and Turner (2005) identified barriers, both perceived and actual, that must be considered when introducing agile processes in the place of traditional ones. These barriers are divided into development process conflicts, business process conflicts and people conflicts. When discussing development process conflicts the authors mention that applying agile processes to legacy systems might not always be easy. For example, refactoring, which is a key practice in agile development, is usually very difficult in legacy systems (Boehm and Turner 2005). Among business process issues the authors mention issues connected with human resources (e.g. different positions and different competence may be needed), project progress measurement (e.g. traditional milestones may not always be applicable for agile development), and process standard ratings (e.g. introducing agile processes may affect the rating of a company with respect to certain standards, like ISO or CMMI) (Boehm and Turner 2005). Finally, when investigating people conflicts, the authors (Boehm and Turner 2005) mention issues connected with change management (e.g. usually there is some resistance to change among the staff) as well as some logistical issues (e.g. many agile practices require teams to be colocated) (Boehm and Turner 2005).

4. METHOD

The goal of this study was to perform an early evaluation of SD applicability for replacing the traditional development process currently used at Ericsson (Section 2.1 for information concerning the current development practices at Ericsson and Section 2.2 for the description of SD). The main analysis method used in this study was FFA, which is an analysis method used in early evaluations of change suggestions (Johnson *et al*. 2005). The FFA method is described in more detail in Section 4.1. The information regarding the impact of SD introduction was collected by performing a number of interviews with representatives of those roles in the company that will be most affected by introducing SD. The findings from the interviews were later postprocessed and structured using FFA by the researchers performing this study. The reason for choosing FFA was that the categories included in FFA simply matched the objectives of the study well (better than any other candidate). More detailed information regarding the data collection and analysis can be found in Section 4.2.

4.1. Force Field Analysis

FFA is a method for identifying issues that should be taken into account when deciding whether to implement a strategic change (Johnson *et al*. 2005). FFA is performed by identifying and categorizing data about a change according to the following key factors (Johnson *et al*. 2005):

- *Pushing factors*: aspects of the current situation that would aid implementation of the change (e.g. enthusiastic staff)
- *Resisting factors*: aspects of the current situation that would resist the implementation of the change (e.g. lack of management support)
- *New additions*: additions that must be in place to make the change possible (e.g. new tools must be acquired)

By balancing the Pushing factors and the New Additions with the Resisting factors (as presented in Figure 3), FFA makes it possible to evaluate how successful the implementation of the change is likely to be.

In the study presented in this article, FFA had to be modified slightly since the aim with the study was not to evaluate the probability of success of the change implementation but rather the change (i.e. the SD introduction) itself. Therefore, the definitions

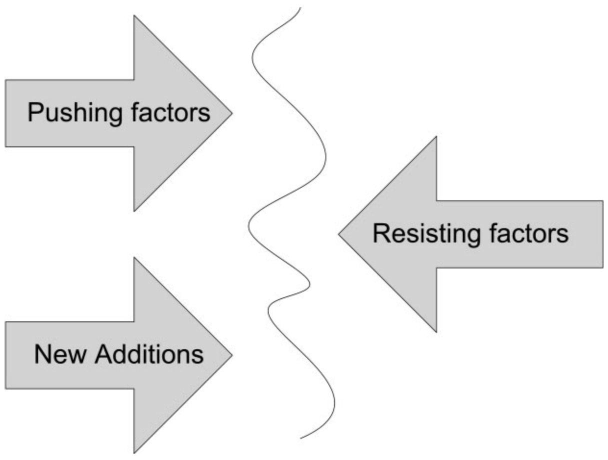

Figure 3. Force field analysis 79 × 59 mm (600 × 600 DPI)

of the three factors taken into account in FFA were slightly modified:

- *Pushing factors*: the advantages of SD. Things that would improve after introducing SD (e.g. customer responsiveness would improve).
- *Resisting factors*: threats connected with introducing SD, i.e. things that would worsen by introducing SD (e.g. the quality of the architecture would deteriorate).
- *Required changes*: Issues that must be resolved and problems that must be overcome before the SD can be implemented (e.g. new competence must be acquired).

By balancing *Pushing* and *Resisting factors* it is possible to make an informed decision if the change is worth introducing or not. The information about the *Required changes* makes it possible to assess the cost of introducing the new processes and to identify issues that must be resolved before the new process can be introduced.

4.2. Data Collection and Analysis

Before the study, a meeting with one manager from each of the two PDUs at Ericsson was arranged. One of these managers was responsible for the evaluation and potential implementation of SD at his PDU, while the other manager was a representative from the management team at the other PDU. During this meeting it was decided that the opinions regarding benefits and drawbacks of SD, as well as opinions about changes that are required for successful implementation of SD, should be collected by performing interviews. Further, the following questions were addressed during the meeting to clarify the scope of the evaluation:

- *From which perspectives should the evaluation be done?* – a number of different evaluation perspectives can potentially be identified, e.g. product perspective, organization perspective but also some role-related perspective like development or marketing perspective. The reason for discussing this issue was to find out which perspectives are of interest for people making decisions about introducing SD.
- *Who should be involved in the evaluation?* – it was important to identify roles that should be interviewed at each PDU in order to capture the aspects of interest. Persons representing these roles were the main source of information in this study.

During the meeting the following perspectives were found important by the managers:

- *Organizational perspective* – when introducing a new development process (such as SD), it is common that the organization must change. This means that it is not only the size of the project, the order of things done, etc. that must change but also work responsibilities, resource utilization, and so forth. Further, it might be so that SD is more applicable for some PDUs and less for others. To capture these issues the interviews were performed at two different PDUs and questions regarding applicability of SD for each of them were asked.
- *Product perspective* – products may have to change due to the different way of developing the software. Since Ericsson has a product portfolio with products with different characteristics, potentially some of these products are more suited than others for development with SD. To capture such issues, questions concerning the applicability of SD for different products were included.

The managers suggested interviewing the major roles that would be affected by introducing SD since this was the only way to provide the holistic picture of the change. In addition, it was seen as important to have equal representation from the two PDUs. The following roles were recommended to be included in the interviews:

- *Operative Product Managers* – persons that can be considered as customers' spokespersons in a project. Their major role is to decide what should be done to meet customer needs.
- *System Managers* – persons responsible for refining requirements and deciding how the requirements can be implemented in the system.
- *Designers* – persons responsible for the designing and implementing the system.
- *Testers* – persons responsible for verifying that the product works properly and according to its specification (Pfleeger 2001).
- *Project Managers* – persons holding the overall responsibility for a project. Project Managers plan, direct, and integrate the work efforts in

order to achieve the project's goals (Nicholas 2001).

- *Configuration Managers* – persons responsible for build environments, product and document storage systems, creation of deliverable products from developed code and documents, and for managing changes.

On the basis of this selection of roles, managers for the different departments where the roles were located were contacted. These managers got instructions to select representative roles from their departments, based on the availability of personnel. In total, 12 interviews were performed (six roles from each of the two PDUs) during which 27 persons were interviewed (approximately 5% of all possible respondents). Each interview was scheduled for 2 h. At each of these interviews, the following persons participated:

- persons representing one role at one of the PDUs (usually two to three persons)
- two researchers, one led the interview and one acted as a secretary

By using this setting, it was possible to have discussions specific to the role and the PDU at the same time. Further, by having multiple representatives for each role, it was possible to get discussions between the interviewees. The discussions were further facilitated by the fact that the interviewees were familiar with each other. By having two researchers present, it was possible to keep a good flow of the conversations at the same time as the necessary information was collected.

As the study was meant as exploratory, a certain degree of freedom was given to the interviewers to enable follow-up questions when some new issues popped-up, to reformulate some of the questions, or to change the order in which questions were asked. Hence, the interviews can be classified as semi-structured (Robson 2002). Another option was to use unstructured interviews, which are very common in exploratory studies (Robson 2002). However, at the time of the study the topic was largely discussed in the company and some people were very opinionated. By using purely unstructured interviews, there could be a risk that the conversations would focus only on positive or negative aspects of the case. This could have led to collecting only partial information about the studied situation. By adding some structure to the interviews this risk was reduced. To further reduce this risk the questions asked in the interviews were deliberately formulated in a way that would force interviewees to focus both on positive and negative sides of each issue discussed (e.g. 'Why will not SD work in relation to your particular product?' but also 'Why will SD improve the product?'). After the structure of the interviews (including questions) was developed, it was validated and improved based on the input from the two responsible managers. In addition, those respondents that did not have a sufficient basic understanding of SD, were given education on the concept before the interview started to ensure that all respondents had a sufficient basic understanding of SD. However, it should be noted that even though the respondents had a sufficient understanding, and perceived SD similarly, they still had different opinions/perceptions about the result of the introduction (e.g. some thought that maintenance will be simpler, and some thought it will be more costly, see Section 5).

After each interview, the secretary postprocessed the information collected and prepared the final result of the interview. When postprocessing the information, the secretary validated the result and potential repetitions and ambiguities were removed. As a final validation, the person leading the interview went through the result and discussed potential discrepancies in views with the secretary. If any discrepancies were found, the issue was discussed until consensus was reached. To facilitate the analysis, the statements describing the opinions of the interviewees were collected in a spreadsheet. This spreadsheet was later used to make the classification of the data easier in the analysis part explained below.

The final grouping of all the identified factors according to FFA (i.e. into Pushing factors, Resisting factors, and Required changes) was done jointly by the researchers involved in the study during a consensus meeting. However, some problems were experienced when doing this. First of all, it was problematic to get a good picture of the factors due to the size of the data set (about 300 statements representing opinions of the interviewees). Another problem was that the opinions were stated at different levels of abstraction, which made them hard to compare. To address these problems, similar statements were grouped and presented on a comparable abstraction level by focusing on keywords (inspired by content analysis Robson

2002). When doing the analysis, the major purpose was to collect all the statements, put similar ones into groups, categorize them according to FFA, and to present the data in a usable form so that the findings can be useful for the decision makers at Ericsson. It should be noted that this activity was done jointly by the researchers, and that the categorization was done by striving for a consensus between the researchers. Hence, all three researchers agreed on the classification of the statements.

5. RESULTS

This section presents the most important results of the study. It must be clear that the statements presented in this section are based on the opinions of the interviewees about the potential introduction and usage of SD, i.e. they have no actual experience with SD as SD has not yet been implemented. The results presented in this section represent the main findings and are presented according to the classification presented in Section 4.1, i.e. as Pushing factors (Section 5.1), Resisting factors (Section 5.2), and Required changes (Section 5.3). The main findings presented are those findings that were considered by most respondents and/or those that were seen as extremely important by the respondents.

5.1. Pushing Factors

The Pushing factors are the positive effects of introducing SD. They are summarized in Table 1.

A common view among the interviewees was that introducing SD would result in *increased motivation of the staff* involved in the software development. As major reason for that, the interviewees mentioned the positive impact of short projects. In short projects, the end of a project is more ‘tangible’ and thereby people will be more motivated. Another motivating aspect of short projects, according to the interviewees, is that short projects provide almost immediate feedback and make it possible to quickly see the results of their own work.

The interviewees shared an opinion that smaller projects will lead to *higher productivity*. The reason for increased productivity was the fact that short projects are less exposed to the risk of changing requirements. The change of requirements commonly involves waste because already performed work has to be remade (or removed) to meet changed requirements. According to the interviewees, the more stable scope in small projects minimizes the risk of waste (and hence unnecessary work) and therefore, leads to higher productivity.

Interviewees also mentioned that SD can give a competitive advantage, because it will *increase the customer responsiveness* by delivering the products to the customers fast. The short time between ‘ordering’ functionality and getting it should be very attractive from Ericsson’s customers’ perspective. The customers are interested in getting new systems with new features fast because these systems often give them a competitive advantage.

Another positive aspect of introducing SD is connected with the idea of producing only one version of the system and incorporating customer adaptations into the main product. The interviewees shared the opinion that this should *simplify the maintenance* and reduce its cost. Currently, many product branches have to be maintained (e.g. for each customer adaptation), which is very expensive.

SD should also *improve the communication* within the projects. The development teams will be smaller and the communication within the teams will be easier and more frequent. Therefore, relatively more time can be spent on producing the system than on controlling and synchronizing development team members. According to the interviewees, the risk of costly misunderstandings (e.g. misunderstandings concerning requirements) will decrease since the collaboration between roles will be closer as a result of improved communication.

Owing to the improved communication within projects and closer cooperation between different roles, the overall *competence level should increase* since team members will have more chances to learn from each other. The learning process will be further facilitated by small projects. In small projects the time between requirement specification, implementation and testing will be short, which will provide instant feedback to all roles. This will promote and facilitate personal improvement; the competence of individuals should increase because of quick feedback on the quality of their work. Additionally, the projects will be more frequent and the personnel will do their tasks more often. Thus the overall competence related to performing a project should also increase.

The respondents also believed that SD should *improve the controllability* of the projects, mainly

Table 1. Summary of Pushing factors of SD

Factor	Description
Increased motivation of the staff	Shorter projects and immediate feedback will let everyone involved in the development process see the results of their work faster.
Improved productivity	Shorter projects tend to have more stable scope because they are less exposed to the risk of changing market demands. Stable scope minimizes the waste since not that much rework must be done to adapt the system to new requirements.
Increased customer responsiveness	SD makes it possible to release new versions of the system more frequently and hence respond to new customer demands quickly.
Simplified maintenance	In SD there will be only one version of the system produced with no branching for customer adaptations. That will minimize the number of maintained system versions and, therefore, minimize the cost of maintenance.
Improved communication	Projects will be performed by smaller teams. Small team size makes it possible to have more frequent and efficient communication between all team members.
Increased competence level	Fast feedback will facilitate personal development. Project-related competence will improve due to frequent projects.
Improved controllability of the project	Due to smaller project size and scope it will be easier to control the project but also to perform estimations and predictions concerning the project.

because of the reduced size of the projects and better communication within them. Small projects make it possible to make more correct predictions and estimations regarding their lead-time and cost. It is also easier to obtain and maintain an overall picture of what is happening in the project and, therefore, it is also easier to monitor progress.

5.2. Resisting Factors

The Resisting factors are the effects of introducing SD that, according to the interviewees, will have a negative impact on software development at Ericsson. They are summarized in Table 2.

The interviewees often mentioned *problems with assuring the quality* of the systems as possible drawbacks if introducing SD. Since SD puts a lot of pressure on keeping the projects short, the interviewees saw a risk that quality assurance may suffer. The interviewees also perceived SD as a very 'feature oriented' process – the small projects' primary goal will be to deliver the functionality as quickly as possible. This attitude may promote short term thinking because of which long term goals, like maintaining the quality of the systems, may suffer.

According to the interviewees, one of the major undesired consequences of the 'feature orientation' of SD may be *architecture deterioration.* Since SD is feature oriented, architectural improvements may be neglected when planning which requirements to implement thus causing architecture deterioration. Another problem related to architecture deterioration is that small projects are not appropriate for implementing large architectural improvements. This may make it hard to address architectural deterioration problem after introducing SD.

Some interviewees mentioned *high maintenance cost* as one of the problems. This may sound like a paradox, because maintenance cost decrease was mentioned as one of the positive aspects of introducing SD (Section 5.1 for details). However, some interviewees noticed that another feature of SD, the ability to release very often, may lead to a situation that there will be a large number of system releases present in the market. Today a new version of the system is available, on average, every year and a half. With 3 months projects, that additionally can be overlapping, a new version of a system can appear much more often. Since customers are not always willing to update their systems so frequently, there is a clear risk of having a large number of releases out in the market that have to be maintained and supported.

Table 2. The summary of Resisting factors of SD

Factor	Description
Difficulty in assuring quality	SD suggests keeping projects short. Quality assurance activities often tend to suffer when cutting costs in the project.
Long-term architecture deterioration	The main focus of short projects is on adding functionality in an efficient manner. That causes a risk of taking short cuts and forgetting about issues like system architecture. Furthermore, such architecture deterioration may be hard to address in SD as large architectural changes will be hard to introduce in small projects.
Increased maintenance cost	Frequent releases and lack of possibility to enforce their installation on customers causes the risk of having a number of releases in the market that must be maintained.
Increased backward compatibility cost	Doing frequent releases means that each new release must support data migration from a large number of previously released systems.
Dependence on individuals	In small projects the dependence on the skills and knowledge of individuals is large – there is a high risk of losing competence when losing a team member.
Applicability to legacy systems	SD must be supported by the architecture of a system. Legacy systems were not architected with SD in mind.
Old (legacy) processes are still used	Even though SD requires rather dramatic changes of current processes, there might be a risk that personnel is used to old ways of working, and continues working in the old way with only slight modifications.

Another problem with having many releases of systems present in the market is the need of keeping *backward compatibility* when releasing a new system version. For example, it happens that the database format changes between different releases. In such a case, an installation program of the new release of the system has to perform data migration to the new data format. Having many releases on the market would mean that an installation program would need to support data migration from a large number of database formats.

The interviewees shared an opinion that the *dependence on individual persons* is a much larger problem in small projects compared to the current large ones. The reason for that is that developer teams will be smaller in SD and therefore, the consequences of someone dropping out from a project are likely to be more severe. Such a loss would be very difficult to compensate within the time frame of a short project.

Some of the interviewees were concerned about the *applicability of SD to already existing, large software systems*. They argue that the system architecture must support software development in a number of small concurrent projects. Also the dependencies within the system must be well understood to enable efficient project planning (projects should not collide with each other). Such good knowledge of system structure is often a problem in the case of legacy systems. Additionally, some SD concepts also require certain support from the architecture. One example can be the idea of integrating customer adaptations with the main product. Such adaptations are usually made for one particular customer and should not be accessible for other customers. This implies that the architecture should provide means of switching on/off certain functionalities. Since legacy systems were not architected with this SD specific requirements in mind, there is a risk that their architecture is not suitable for SD.

Apart from legacy systems the respondents also discussed *legacy processes*. When changing to a new way of working, new processes that will fit this new way of working must be developed as a replacement for the old processes. Even though everyone agreed that this is the way to introduce the change, several interviewees saw a clear risk that it will be tempting to try to use currently existing processes in SD. The old processes are not meant to be used in an environment like with SD, which means that the usage of old processes may affect SD negatively.

5.3. Required Changes

Required changes are issues that, according to the interviewees, must be tackled before introducing SD if the introduction of SD should be successful. The Required changes are summarized in Table 3.

When discussing the required changes connected with introducing SD, most of the interviewees stressed the need of introducing *continuous requirements management*. This means that requirements must be packaged and prioritized so that it is always clear what a new small project should do. Packaging requirements means that it is necessary to find chunks of requirements that fit well together and, at the same time, are of suitable size to implement in the small projects. When having such packages, it is important to continuously prioritize these in order to always choose the requirements packages most suitable for implementation (considering dependencies, cost, market window, etc.). This activity must be continuous as new requirements are continuously incoming. Additionally, since requirements are usually prioritized towards the current baseline (the characteristics of the current system version) their prioritization should be updated every time the baseline changes (i.e. when some new project is integrated) and the prioritization practices must be able to cope with such situations efficiently. Continuously prioritizing the requirements is important because changing the baseline may actually change the importance of a requirement.

The interviewees also pointed out that the *effectiveness of pre-project planning must be assured*. When planning a new project it is necessary to assure that there will be no clashes with other ongoing projects. This planning must also take into account that some decisions must also be made upfront, before the project starts. For example, if the project requires an access to a customer site (e.g. for testing purposes) then appropriate arrangements usually must be made well before starting the project. This improvement is of course highly interconnected with the discussion in the previous paragraph since project planning issues must be taken into account when prioritizing requirements and planning releases. At the same time, results from prioritizations must be taken into account when planning the projects.

SD enables frequent releases of the new system versions. In large telecommunication systems, it will (according to the interviewees) be very important to assure an *efficient installation procedure*. If the deployment on customer site is to be done more often, an inefficient installation procedure may become a major cost. SD will also require frequent installations in the test plant. Therefore, an efficient installation will also facilitate and decrease the cost connected with testing of the systems.

In general, the respondents found it very important to *increase testing efficiency*. They considered testing efficiency as one of the possible productivity bottlenecks in SD. One major reason for that is that small and frequent projects will mean that testing will be done more often but there will be shorter time to do it. Since there is some overhead connected with testing (e.g. regression testing must be repeated for each system increment) the impact of testing on project cost is likely to increase. Therefore, the requirements for testing efficiency will be much higher in SD and automatic tests must be utilized to a larger extent.

Since individual capabilities are likely to play larger role in small projects (Section 5.2), the interviewees found it very important to *keep people in the same product line*. Moving people continuously between different products may result in that their learning process will affect the overall project productivity negatively. One of the goals with SD is to increase the overall productivity of software development by having it done in small, efficient, and specialized teams of developers. This advantage may be lost if people will be continuously moved from one team to another, as is currently done.

The interviewees also found it important to fully *understand the dependencies between system components*. They suggested creating an anatomy plan (Taxén and Lilliesköld 2005) in which the interdependencies between different system components would be described. Without the knowledge about these interdependencies it is impossible to coordinate a number of concurrent projects and ensure that they will not collide with each other. The knowledge about interdependencies in the system is also necessary for requirements prioritization and packaging as well as pre-project planning, i.e. requirements affecting the same parts of the system should in most cases be implemented together.

It is also very important to focus on *architecture improvements* before implementing SD. Two reasons why this issue is important were identified.

Table 3. The summary of Required changes connected with introducing SD

Suggestion	Description
Introduce continuous requirements management	The inflow of new requirements is constant. The requirements must be continuously prioritized to facilitate release and project planning. Also the requirements prioritization depends on the current baseline (system version). Therefore it must be updated every time the baseline changes, i.e. every time a new system version is released.
Assure effective pre-project planning	A number of concurrent projects must be coordinated so that there will be no collisions between them. Also many things (e.g. allocation of some resources) that can now be planned in the project will have to be planned before project starts.
Increase the efficiency of the installation procedure	Efficient installation procedure will make testing more efficient. Also frequent releasing and deployment will make the efficiency of the installation process more important.
Increase testing efficiency	Testing is likely to be one of the productivity bottlenecks in SD. Therefore new methods for improving the efficiency of testing will be necessary.
Keep people in the same product line	It will be important not to move people between different products as in short projects the learning effect will have large impact on productivity.
Assure understanding of dependencies within the systems	To enable coordination of small projects it is important to fully understand the dependencies between all components of the system. That will make it possible to avoid collisions between projects.
Improve architecture	Some required architecture improvement work needs to be performed before SD is introduced to make the system architecture fit SD. It also seems that it will be harder to address large issues like architecture changes within SD.

First, one of the common opinions of our interviewees was that SD requires support from the system architecture. The architecture of the system must make it possible to perform concurrent development projects and maintain a single system version. A second reason why architectural issues are important to address before implementing SD is that it apparently will be very hard to address these issues after SD is introduced. ‘Feature orientation’ and limited scope of short projects, together with the necessity of maintaining one product version only, will make it very difficult to address some large and fundamental issues, like for example architecture change or improvement.

6. DISCUSSION

In this section, the results of the study are discussed in more detail. Section 6.1 focuses on the discussion of the major findings and Section 6.2 compare these findings with findings of other researchers. Finally, validity issues are presented in Section 6.3.

6.1. Discussion Regarding the Results

The goal of the study was to evaluate the applicability of SD at Ericsson. Unfortunately, all details of this evaluation cannot be revealed due to confidentiality reasons. However, as can be noticed in the discussions in previous sections, the overall attitude towards SD was positive as there was a rather large agreement between the interviewees when it comes to the positive effects of introducing SD. These positive effects were very similar to the intentions of SD (Section 2.2), which indicates that SD is likely to achieve the goals it was designed to achieve (e.g. improved customer responsiveness and productivity). The fact that the interviewees were positive about the new process is a good forecast for the success of the SD implementation as staff’s resistance to change is often considered a problem when new processes are introduced (Boehm and Turner 2005). Additionally, by looking at *Resisting factors* and *Required changes*, it can be seen that many of the *Required changes* actually either address or at least reduce some of the problems classified as *Resisting factors*. For example, insufficient knowledge about legacy systems and their architecture deficiencies were considered as one problem with applying SD to legacy systems (Section 5.2 for details). By suggesting an anatomy plan and architecture improvement before introducing SD, the interviewees addressed this problem and minimized its impact. Therefore, it seems that by taking some preventive actions the impact of *Resisting factors* can be reduced.

The analysis presented in this paper can be considered an early evaluation or the first step in the evaluation of SD. The main focus was to identify and classify factors that should be taken into account when making the final decision regarding the introduction of SD. Some general conclusions regarding the usefulness of SD can be drawn from this study, but they are, however, by no means sufficient for making the final decision. Nevertheless, the study was considered useful from an SD evaluation perspective by decision makers at Ericsson. It made it possible to check if, in general terms, SD seems promising and if there are no obvious obstacles for introducing it at Ericsson. It was also a way of spreading knowledge about SD in the organization, at the same time as employees got the opportunity to express their opinions about SD (which probably increases the chances of success). Based on this information, it was possible to decide if it makes sense to put more time and effort into further development and evaluation of the SD idea. The study was conducted using FFA approach, and one of the good things about performing evaluations with this approach (Section 4 for details regarding the method) is the limited amount of time necessary to perform it. For the case study presented in this paper, it took about two working weeks for three persons to perform interviews, collect and analyze the data, and present the findings in a usable form to the decision makers. Therefore, the low cost combined with the effectiveness makes this evaluation method a useful tool in early evaluations of new development processes.

The next step in the evaluation of SD is naturally to perform a detailed cost-benefit analysis of introducing SD, in which the impact of each of the *Pushing factors*, *Resisting factors*, and *Required changes* should be quantified. By balancing the benefits from introducing SD (i.e. *Pushing factors*) with the costs associated with risks (i.e. *Resisting factors*) and changes (i.e. *Required changes*) it will be possible to make the final decision regarding the introduction of SD. As it can be noticed, further analysis can be still performed in the frame of FFA. This indicates that FFA can be a very helpful analysis method in evaluations of different change suggestions. In this study it has successfully been used in process change evaluation but it seems to be also applicable for an evaluation of any other changes e.g. technology change.

6.2. Comparison with Other Studies

As indicated in Section 3 SD has many similarities with many other modern software development practices. Therefore, it is interesting to compare the results obtained in this study with the findings of other researchers that evaluated such modern development processes and compared them with more traditional ones. Not surprisingly, like many others (e.g. Tran and Galka 1991, Jensen 1995, Benediktsson and Dalcher 2003) the interviewees in this study also observed the positive impact of increased customer responsiveness obtained by introducing practices like SD. Similarly to Graham (1992) the interviewees also noticed that the cost connected with managing CRs should be reduced, mainly because the number of CRs should decrease due to shorter projects. Another observation from Graham (1992) that was confirmed is the positive impact of shorter projects on the motivation of the staff. A common conclusion in many studies is that all these positive effects lead to higher productivity and shorter lead time (Jensen 1995, Dalcher *et al*. 2005). The majority of our interviewees expressed that they expect this also from SD. Also the negative effects of introducing SD described in this study have been recognized in other studies. For example, coordination problems and requirements prioritization challenges are mentioned in (Graham 1992). Another problem recognized by other researchers is the problem with applying new processes to legacy systems (Boehm and Turner 2005), which was also identified in this study. The difficulty to implement large and complex features in small projects, mentioned by the interviewees, was also recognized in (Niels 2001). The fact that our findings are, to a large extent, overlapping with the findings of other researchers, increases the external validity of the identified factors.

There are also findings reported by other researchers that were not fully confirmed in this study. For example, many studies (Jensen 1995, Middleton 2001, Niels 2001) discuss the positive impact of modern practices on system's quality. In this study, on the other hand, the interviewees at Ericsson were concerned about some quality issues, e.g. they stressed that the 'feature orientation' of SD may lead to the degradation of the architecture. However, the differences may be explained by the fact that quality can be understood in different ways (Kitchenham and Pfleeger 1996). In (Jensen 1995) the quality is defined in terms of cost and customer satisfaction, which our interviewees also expected to improve. In (Middleton 2001, Niels 2001) the authors define quality in terms of defects. This quality view, i.e. the impact of SD on the number of defects in systems, was not mentioned by the interviewees and it is hence impossible to say if this finding was verified or not.

Another issue reported by other researchers (Graham 1992, Benediktsson and Dalcher 2003) that is not fully supported by our findings, is the reduction of maintenance cost. The interviewees of this study argued that releasing the software frequently, which is necessary to achieve customer responsiveness, can have negative impact on the maintenance cost. This is because frequent releases may result in many different system versions on the market that have to be maintained (Section 5.2 for details). On the other hand, some interviewees actually agreed that maintenance costs would decrease because only one branch of the product will exist. Therefore, since it is hard to say which of these two issues will have a predominant impact on the cost of maintenance, it is not possible to determine if the maintenance cost will increase or decrease as a result of introducing SD.

It can be observed that most studies in the area (Section 3 for examples) focus on a situation, in which a company can simply choose between traditional and some new development method. In practice, however, organizations most often have a traditional process already in place. Such perspective may change the importance of different factors. It also adds a new angle, the change implementation process, which normally is not accounted for when simply comparing two different development processes. At the same time, it is an important factor in making an informed decision regarding a process change. In this study, this factor was taken into account (in FFA this factor is represented by *Required changes*) as can be seen for example by the important issue of legacy processes (Section 5.3). This is one of the strengths of the presented study.

6.3. Validity

This case study was performed in a concrete industrial setting. Therefore, it may seem hard to generalize the findings to other contexts. However, the actual findings from this study, i.e. *Pushing factors, Resisting factors*, and *Required changes*, may be of interest for other companies that consider moving from a traditional to a modern approach with shorter projects and more frequent releases. For example, it is highly likely that large fundamental changes, like architectural changes, will be hard to address in small projects. Also things like continuous management of requirements must always be in place in order to make the best use of short projects and to reach a high level of customer responsiveness. Even though the individual issues are likely to be of different importance for different companies, they can still help them when identifying threats, opportunities and costs of changing the development process. Also, it seems that the method used for evaluation (FFA, Section 4) seems to be useful in evaluations of new development processes. Before using FFA in the large scale analysis, the FFA categories were tested on samples of data to assess their reliability and the usefulness of FFA (as suggested in content analysis Robson 2002). Therefore, the usage of FFA in this context can be seen as one of the contributions of this article.

Another problem with this kind of studies is that statements, opinions, judgments, etc. may be misinterpreted. However, all three researchers have a long history of collaboration with Ericsson which means that all three are knowledgeable in processes, company culture, etc. This fact reduces the threat that things have been misinterpreted and hence increases the likelihood that the correct issues actually are reported. This experience from Ericsson was also very helpful when performing interviews since the previous knowledge and experiences made conversations easier and more effective. It should also be noted that even though the results of this study aligns very well with previous research (see 6.2), the interviewees were not steered by such results. During the interviews, the discussions were totally handed over to the interviewees and all findings made are directly traceable to personal opinions of the interviewees.

7. CONCLUSIONS

The goal of this study was to perform an early evaluation of SDs applicability as a replacement of the current development practices at Ericsson. SD is a new process created for and by Ericsson with the main purpose of improving customer responsiveness. The evaluation was conducted by interviewing 27 persons from two PDUs at Ericsson. The interviewees represented roles in the company that will be affected by changing the development process. To analyze the findings from these interviews, an adaptation of the FFA method was used. In this method all the opinions were classified as *Pushing factors* (issues that would improve if a new process was introduced), *Resisting factors* (issues that would worsen if a new process was introduced), or *Required changes* (changes that must be made to prepare organization and products for a new development process).

The overall conclusion from the study was that SD seems promising. Its main goals (i.e. improvement of customer responsiveness and productivity) were recognized as *Pushing factors*, which indicates that they are likely to be achieved. On the other hand, some issues were identified and classified as *Resisting factors*. However, many of these were addressed by suggestions of *Required changes*. This indicates that certain actions can be taken in order to decrease the negative impact of those *Resisting factors*.

An additional conclusion from this study concerned the method used to evaluate SD (i.e. collect the opinions regarding the new process by performing interviews and structure them using FFA). This method was found to be a very useful tool for evaluating the applicability of SD. It provided information that was considered valuable by decision makers at Ericsson. Due to the relatively low cost of performing such evaluation, it seems to be especially useful as a method for early evaluations of new process ideas.

The results of the investigation presented in this paper influenced the decision makers at Ericsson regarding if and how to implement SD. The decision was made to introduce SD, and after it is fully implemented, a future study should evaluate the result of the actual implementation.

ACKNOWLEDGEMENTS

We would like to thank Wayne Strong for guiding us through the world of strategy evaluations. We would also like to thank the managers and the interviewees at Ericsson for their active participation in the study. This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project 'Blekinge – Engineering Software Qualities (BESQ)' (http://www.bth.se/besq).

REFERENCES

Beck K, Andres C, Safari Tech Books Online. *Extreme Programming Explained Embrace Change*. Addison-Wesley: Boston, MA, 2005.

Benediktsson O, Dalcher D. 2003. Effort estimation in incremental software development. *IEE Proceedings-Software* **150**(6): 351–358.

Boehm B, Turner R. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE Software* **22**(5): 30–40.

Cockburn A. 2002. *Agile Software Development*. Addison-Wesley: Boston, MA.

Dalcher D, Benediktsson O, Thorbergsson H. 2005. Development life cycle management: a multiproject experiment. *12th IEEE International Conference on the Engineering of Computer-Based Systems, 2005, Green belt, MD, USA*, 289–296.

Graham DR. 1992. Incremental development and delivery for large software systems. *IEE Colloquium on Software Prototyping and Evolutionary Development*, London, UK 2/1–2/9.

Jensen BD. 1995. A software reliability engineering success story. *AT&T's Definity PBX. Proceedings of the Sixth International Symposium on Software Reliability Engineering*, Toulouse, France, 338–343.

Johnson G, Scholes K, Whittington R. 2005. *Exploring Corporate Strategy*. Financial Times Prentice Hall: Harlow.

Kitchenham B, Pfleeger SL. 1996. Software quality: the elusive target. *IEEE Software* **13**(1): 12–22.

MacCormack A, Kemerer CF, Cusumano M, Crandall B. 2003. Trade-offs between productivity and quality in selecting software development practices. *IEEE Software* **20**(5): 78–85.

Middleton P. 2001. Lean software development: two case studies. *Software Quality Journal* **9**(4): 241–252.

Nicholas JM. 2001. *Project Management for Business and Technology: Principles and Practice*. Prentice Hall: Upper Saddle River, NJ, London.

Niels JR. 2001. Putting it all in the trunk: incremental software development in the FreeBSD open source project. *Information Systems Journal* **11**(4): 321–336.

Pfleeger SL. 2001. *Software Engineering: Theory and Practice*. Prentice Hall: Upper Saddle River, NJ.

Pfleeger SL. 2006. *Software Engineering: Theory and Practice*. Pearson/Prentice Hall: Upper Saddle River, NJ.

Poppendieck M, Poppendieck T. 2007. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley: Upper Saddle River, NJ.

Redmill F. 1992. Incremental delivery-not all plain sailing (software development). *IEE Colloquium on Software Prototyping and Evolutionary Development*, London, UK, 6/1–6/6.

Robson C. 2002. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Blackwell Publishers: Oxford, Madden, MA.

Schwaber K. 2004. *Agile Project Management with Scrum*. Microsoft Press: Redmont, WA, USA.

Schwaber K, Beedle M. 2002. *Agile Software Development with Scrum*. Prentice Hall: Upper Saddle River, NJ.

Sommerville I. 2004. *Software Engineering*. Addison-Wesley: Boston, MA.

Taxén L, Lilliesköld J. 2005. Manifesting shared affordances in system development: the system anatomy. *The 3rd International Conference on Action in Language, Organisations and Information Systems*, Limerick, Ireland, 28–47.

Tomaszewski P, Lundberg L. 2005. Software development productivity on a new platform – an industrial case study. *Information and Software Technology* **47**(4): 257–269.

Tomaszewski P, Lundberg L. 2006. The increase of productivity over time – an industrial case study. *Information and Software Technology* **48**(9): 915–927.

Tran P, Galka R. 1991. On incremental delivery with functionality. *Proceedings of the Tenth Annual International Phoenix Conference on Computers and Communications Conference*, Scottsdale, AZ, USA, 369–375.

Wiegers KE. 1999. *Software Requirements*. Microsoft Press: Redmon, WA.

Womack JP, Jones DT, Roos D, Massachusetts Institute of Technology. 1990. *The Machine that Changed the World: Based on the Massachusetts Institute of Technology 5-Million-Dollar 5-Year Study On the Future of the Automobile*. Rawson Associates: New York.