# Process Improvement Solution for Mobile Platform Customer SW Development

by

*Anis Yazidi, Terje Leira*

Supervisors: Ole-Christoffer Granmo, UiA
Ole Dag Svebakk, Ericsson

**IKT 590 Master's Thesis**

**May, 2008**

University of Agder
Faculty of Engineering and Science

**Keywords:** Software, SPI, Change Management, Agile development, Lead time

## Abstract

Time to market is becoming an increasingly important topic in software industry. In this trend, handling customer change requests is of a paramount importance. In the current thesis, we investigate reducing the lead time of handling customer requests at EMP Grimstad. Problems were identified and an extensive solution that covers all the aspects of these problems are presented. An experience was conducted and the first results are promising.

# TABLE OF CONTENTS

# 1 Introduction

The mobile industry is a rapidly changing arena, where time to market is becoming a competitive key. In order to meet the customer requirements, new models have to be released in the nearest future. Besides, novel services have to be integrated at a high pace within the mobile existing systems. Mobile manufacturing companies are competing to shorten their development time along with upgrading the quality and efficiency. The importance of a short development lead time has been emphasised in the literature (Datar et al .1997; Porter, 1980; Wheelwrigt et al., 1995). A short development lead time enables the company to grab market lead and avoids the market lockout (Shilling; 1998). In a survey (Bratthall; 1994) conducted to assess the impact of lead time on the business profit, Bratthall reported a case where a one week delay to the market resulted in a company's total loss of the potential market and thereby a dramatically waste of millions of dollars.

In this master thesis, we address a problem akin to the field of Software Process Improvement (SPI): reducing the lead time of handling software change request (CR) at Ericsson Mobile Platforms (EMP) in Grimstad. EMP are the developers of a generic platform which their customers use in their mobile phone development projects. The EMP department in Grimstad is responsible for development of the data communication part of the software platform and to some extend for related hardware parts. The platform development is done in large development programs and each program spans over a relatively long time period. However, mobile phone manufactures are launching new models more or less every month. In order to meet the mobile phone manufactures requirement/wishes, EMP needs to add new functionality to the platforms in a way that ensures a shortest possible lead time.

Reducing the development lead time is a problem that relates to Software process Improvement (SPI). To this date, a myriad of papers have been published pertaining to improve the existing software practices, quality and subsequently the development speed.

Nevertheless, only a small body of the literature has addressed the reduction lead time of customer change requests (CR). The only few studies that focused on handling change requests emanates from the industrial field.

In this thesis, we try to fill the void in this field by proposing a study that combines approaches from the SPI field with ideas inspired from the industrial field. To achieve such a goal, we put the sate of the practice at Ericsson into critical analysis to discern deficiencies in the CR process. We claim that these deficiencies are related to the nature of the underlying waterfall model. We claim that a solution that combines features from the manufactory domain with features from the software process improvement field will be able to solve the problem.

# 2 Research design

One preliminary consideration before designing a proposal for a solution is to identify a framework for the study.

## 2.1 Research methods

A survey of the literature related to research design discerns three main approaches to research: quantitative, qualitative and mixed methods research (Creswell, 1994). A quantitative approach is where the researcher makes use of strategies of inquiry, such as experiments and surveys, with focus on statistical data. Alternatively, a qualitative approach is where the researcher bases his strategy of inquiry on open-ended questions, interview data, audiovisual data, text and image analysis and develop themes from the data. Finally, a mixed methods approach is where the researcher uses mixed methods design: close ended measures and open ended observations. Thus, in a mixed approach, the data collection involves both quantitative and qualitative information. The three books (Juristo, 2001; Creswell, 1994; Cooper, 2001) provide a good comprehensive in depth survey of research design and methods.

## 2.2 Adopted approach

In our master thesis we plan to proceed with a bottom up approach. A bottom up approach is an inductive research approach where the researcher creates hypotheses and builds underlying theory based on data collection. The premise of such approach, when applied to Software Process Improvement, is that we first should understand the existing problems before improving the process. To identify and understand the problems, we will utilize the experience of the organization.

Being aware that the technical staff knows best the characteristics of the company, our quest for knowledge is centred on interviews. The qualitative part of the study will rely mainly on Ericsson engineers' experience by conducting questionnaires and interviews.

The interviewing technique is regarded as viable method to reveal software process problems.

To highlight the expediency of interviews Kvale states: "The very strength of the interview is its privileged access to the common understanding of subjects, the understanding that provides their worldview and the basis for

their actions"(Steinar, 1996). Interviews were proved to understand what problems are most important.

The interviewees should be selected according to a theoretical sampling. The reason is that a theoretical sample offers wide disparate views (Marshall, 1996). With regards to the small number of the population, i.e ˜170 Ericsson employees, a reasonably small sample size of people would be sufficient. We plan not to exclude novice employees from the interviews because they are usually full of criticism of the practices at the company. Such acute observers of the process can shed light on hidden faces of the problems that are not perceived by other experienced Ericsson employees.

The interviews should contain a combination of structured interview format and standardized open-ended interview using an Interview Guide. The purpose of open-ended questions is to allow the participants to answer in a way that allows us to know how the interviewees perceive a problem. It also gives the opportunity to pursue with in-depth questions to either validate a hypothesis or to get more insight into interesting or unexpected findings. In this way, the open-ended interviews will support our bottom up approach. The latter also gives the opportunity to refine and modify the interview guide to include "lessons learned" in the subsequent interviews.

Besides the interview, another important component of a qualitative research is performing a literature study. Surveying the state of art is of paramount importance. We will start by surveying studies on software process improvement methodologies to gain insights into the domain of software engineering. In parallel to this academic study, we plan to probe into the Ericsson process. In order to gain a fast and large degree of understanding of the latter process, internal documents the internal software process at Ericsson are prime candidates for scrutiny.

So far, we have presented our qualitative methods. Nevertheless, a comprehensive research approach usually contains a mixture of quantitative and qualitative studies to capture the best of both approaches.

In this perspective, we consider to integrate quantitative data collected from errors reports, change requests and similar statistics in order to discern even more inconsistencies in the software process.

## 2.3 Research questions

The general research question in our thesis is to identify the general problems that drive the lead time at Ericsson Mobile Platforms and design a solution that addresses them.

We have formalized the research question by dividing it into four specific questions:

| | |
|---|---|
| R1 | How can we contribute to a shorter lead time without sacrificing quality? |
| R2 | What are the essential change management problems and bottlenecks? |
| R3 | What kind change management process would aid in solving these problems? |
| R4 | How can the proposed process enacted in practice? |

# 3 Context of the study

In this chapter, we present the context of the study.

## 3.1 The Ericsson Context

## 3.2 About the Company

Ericsson is an international telecom company with headquarter in Sweden. It has approximately 75 000 employees at present. Ericsson has a long experience in software development that spans over many years. In a move to excel on the telecom market, producing better software has been a widespread concern at Ericsson. This study has been performed at Ericsson Grimstad branch. Ericsson Grimstad has been involved in developing software for GPRS from 1997 to 2003.

With the advent of third-generation mobile system (3G) technology for mobile phones, the landscape changed considerably for mobile phone manufacturers and operators. Subsequently, Ericsson Grimstad tuned into a supplier of Mobile Platforms. EMP (Ericsson Mobile Platforms) involves two development centres in Basingstoke (UK) and RTP, North Carolina (US).

### 3.2.1 What is EMP?

EMP was established in September 2001 with headquarters and main development centre situated in Lund, Sweden. The research and development (R&D) constitutes the core of Ericsson Mobile Platforms. EMP R&D units are present in Grimstad (Norway), Tokyo (Japan), Shanghai (China), Taipei (Taiwan), Seoul (South Korea). EMP is software and technology supplier to mobile phone manufactures developing devices complying with GPRS, EDGE and WCDMA standards.

The main purpose of Ericsson Mobile Platforms (EMP) is to provide customers with a basic set of hardware and software components in order to facilitate the development of new mobile phones. As mobile phones are becoming more and more complex, it is tedious for a phone manufacturer to develop and test the technology used in 2.5G (GPRS) and 3G (EDGE and WCDMA) phones in-house. In this context, Ericsson grabbed the market and is well positioned as a third party development provider.

The time to market of a phone varies usually from 18 months to 4 years, depending on its underlying complexity. EMP customers can launch phones with a lead time ranging from 9 to 12 months. Moreover, for increment releases

the lead- time is typically 6-8 months. Manufacturers can thus grab the market in few months. Hence, EMP saves significant effort for phone manufacturers when it comes to development and testing.

### 3.2.2 Platform Overview

## Platform services

The platform software consists of different functional domains in a layered architecture.
These functional domains are the following

-Network access services: which provides access to and services for the radio network

-Data communications services: provide support for data communication services such as Bluetooth, USB, IP services.

-Man-machine interface (MMI) services: provide support for user interface devices such as touch-screens, camera etc.

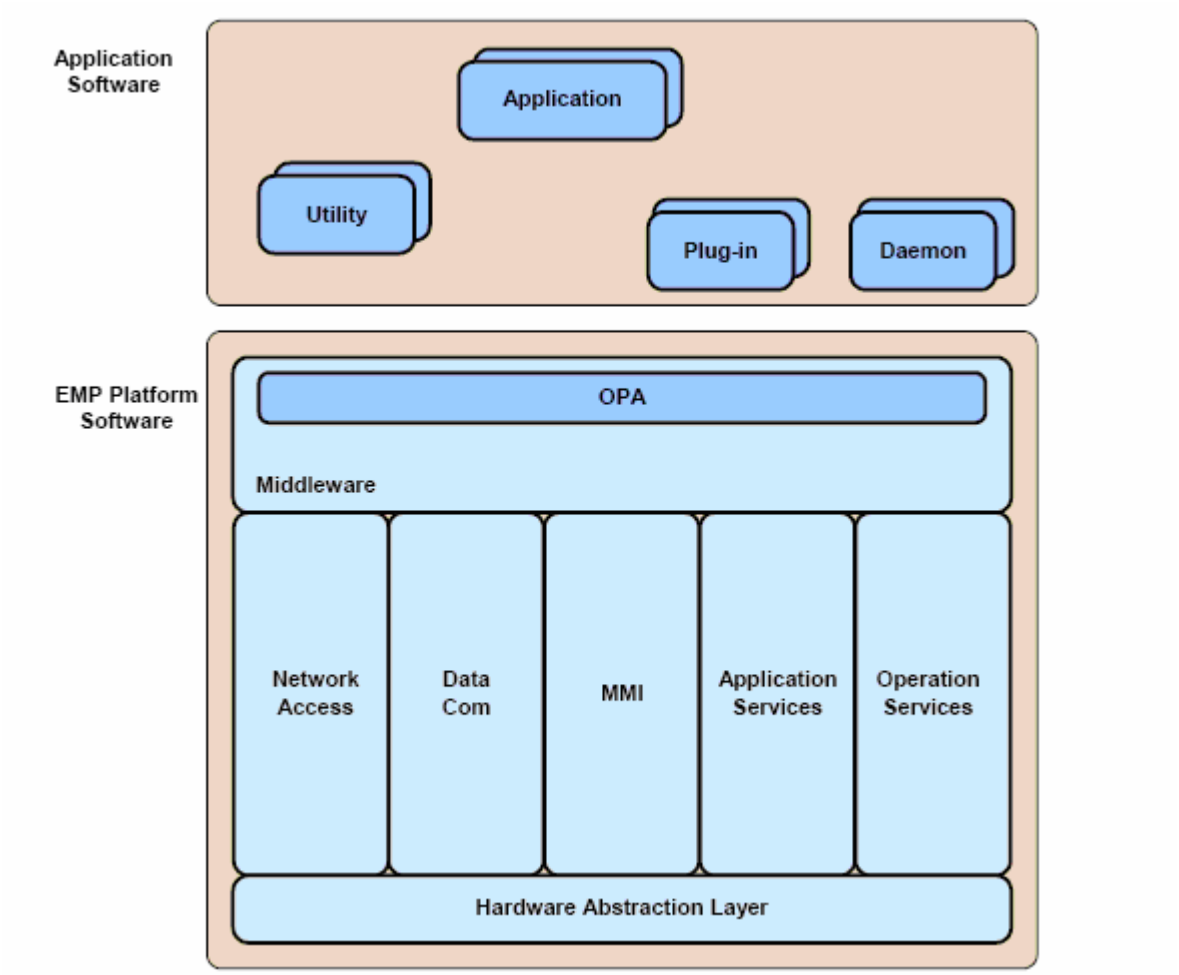
-Application platform services: provide storage support and management.

-Operation services: provide basic operating system, platform startup and shutdown.

The middleware domain defines the application model for the application software. The application model sets the basic rules and defines the environment for the application software. The application model provides the application software with a real-time multitasking environment and a natural way of controlling the application software and using the services of the platform.

All platform functionality is accessed through an extensive API called OPA. OPA is a part of the middleware domain that provides the interface towards the platform functionality. OPA acts as a front-end which provides the interfaces that presents the functionality of the platform to the applications. OPA is designed to reflect the actual services of the platform and is structured in a number of categories where each category or sub-category represents a service.

The mobile platform contains a number of standard services and EMP's customers can develop new applications written in C through a software development kit. This allows customers to create their own unique brand of phones. The figure below depicts the software architecture of EMP.




As seen in this figure the platform can be divided into application software and platform software. All platform functionality is accessed through an extensive application programmers interface OPA.

The system design of the EMP platform is based on use-case analyses. Use-cases are driving the dimensioning of the system. We cite some of theses use cases

- imaging;
- video;
- simultaneous streaming and voice call;
- local and multi-player games;
- synchronization;
- secure access and banking;
- multiple data and voice sessions.

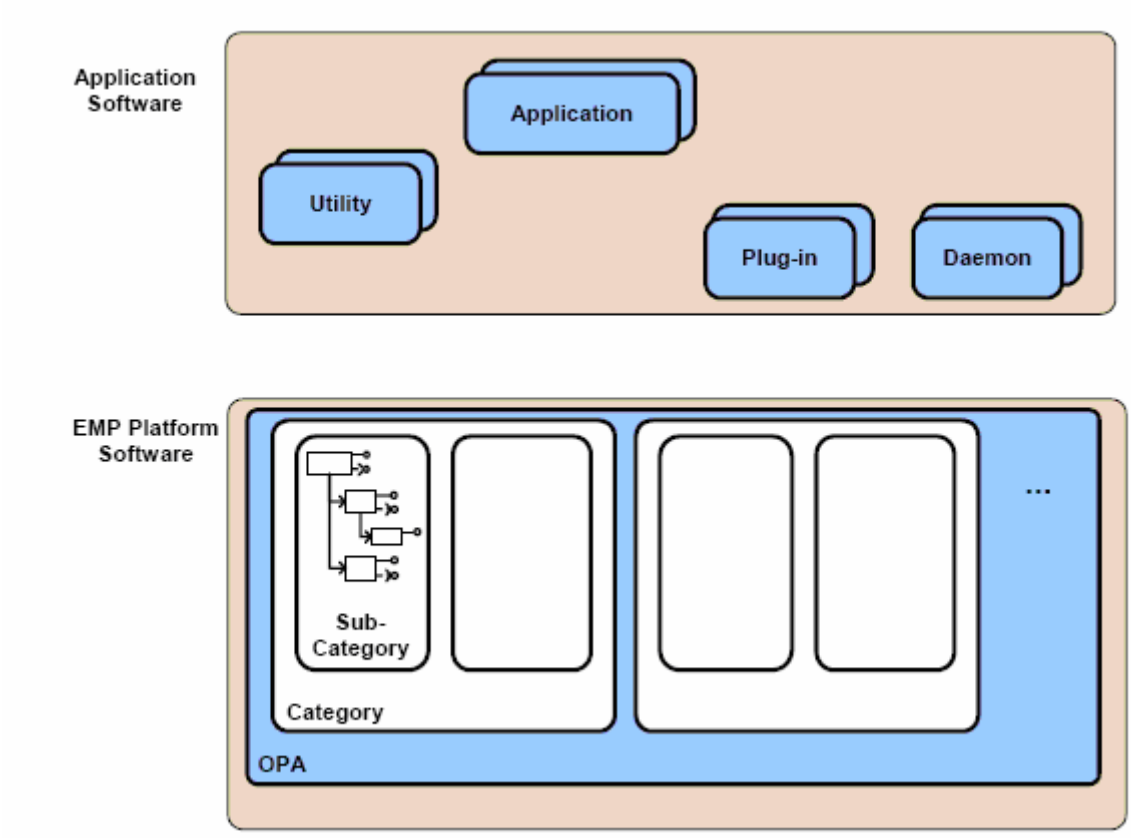
Simultaneous usage scenarios have been the object of scrutiny at EMP, because they usually involve the tight coordination between different hardware such as CPU and DSP-

EMP comprises five service stacks arranged in layers in a similar way to the reference model OSI. The hardware abstraction layer provides a simple device driver interface for programs to communicate with the underlying hardware. The layer of common middleware services interfaces the application software. EMP products are designed for different mobile standards; the common shared feature is the utilization of same global architecture and components.

## OPA

OPA is an easy and efficient interface towards the platform functionality based on a modern, object-based paradigm. OPA eliminates the need for the application developer to have to deal with details in the platform implementation. OPA also reduces specific hardware and operating system dependencies for the application software.

The figure below gives an overview of the structure of OPA and how the OPA services are organized.



As seen in the figure, four different entities can be developed in the application Software

-Application: An applications is a separate executable entity, owning at least one thread.

-Utility: A Utility is a flexible entity that can be used to provide simpler services and to provide adaptation for legacy customer code.

-Plug-in: A Plug-in is used to extend the OPA services while behaving like an OPA service from the client perspective.

-Daemon: A Daemon is a self-standing executable used to extend the platform with services needing a separate thread.

All OPA services are organized in categories and sub-categories in order to obtain a functional structure and partitioning of the OPA services. Each sub-category contains a number of components and these components provide one or more interfaces in order to use the services. The components define an object model for the functionality of this subcategory. Each interface of a component consists of one or more methods.

## 3.3 State of the practice at Ericsson, Grimstad

To get insight into the state of the practice at Ericsson with respect to processes, we have examined internal documents, literature, conducted interviews as well as attended a workshop.

The goal was to get a in-depth understanding of the state of the practice processes with the objective to characterize, describe and analyze the process routines at Ericsson to learn where problems in practice can be found and improved.

### 3.3.1 Background

Ericsson, like most major international companies, run their product development operations as projects. It is known that using project models and processes to manage these projects can make a major improvement of performance:

"Companies that have a consistent approach to managing projects, can perform their projects at a cost 75% less than companies that leave project management practices up to individual project managers" [Eric Verzuh : The Fast Forward MBA in Project Management]

As a tool to facilitate this goal, Ericsson uses a general project management process called Project Management Process System, PROPS. It was originally designed by Semcon AB (Mulder, 1997), and is used by Ericsson units worldwide.

A process is defined as "A set of interrelated or interacting activities which transform inputs to outputs" [ISO 9001:2000]

The competition is the telecommunications industry is hardening, and the demands of the customers force companies to be able to deliver complex technologies, such as the mobile platform, as a turnkey solution. For this reason, it is of crucial importance that the different competence centers within Ericsson have the methodology to work together in "cross-functional" projects.

Managing projects running simultaneously at different Ericsson sites can become a complex management issue. Therefore, PROPS is used at Ericsson to succeed in the increasing challenge of efficient management.

The PROPS methodology was developed with aims to serve as a common methodology for managing projects and to give Ericsson units in different countries a common terminology. Hence, units that are working together on cross functional projects will have a common perspective of the processes.

Another thing of importance, is managing the business aspect. PROPS is also a business decision-making process. It is important that as well as focusing on providing a procedure for better managing the product development projects, that the projects are driven by business objectives. By incorporating defined management checkpoints and tollgates, it keeps focus on where business decisions like continue or cancel are made in a project. Werther or not, is based on an evaluation of the project to determine if it is economically or strategically reasonable to continue. To aid these decisions the model defines documents containing correct and relevant information.

At each gate a business decision is made. A decicion criteria checklist is used to evaluate the results of the preceding activities as well as project status, technical solutions and business issues. The outcome is either to cancel, continue, or continue with alterations to the projects such as changing the scope or the plan.

### 3.3.2 What is PROPS?

PROPS is very similar to the Stage Gate model designed by Cooper (Cooper, 2001). It has a high degree of flexibility when it comes to what technologies to use in the underlying development process.

PROPS is defined on the corporate level and is used to "manage projects by allocating scarce resources, the roles that need to be played by all those involved in the project, the supporting line functions, the criteria to be used for decisions taken inside the project and in relation to other projects, as well as other issues that require a shared view."

PROPS controls the process of product development from the initial prestudy to the conclusion phase with mass production. This the model itself falls into four smaller phases which are:

-Prestudy
-Feasibility
-Execution
-Conclusion

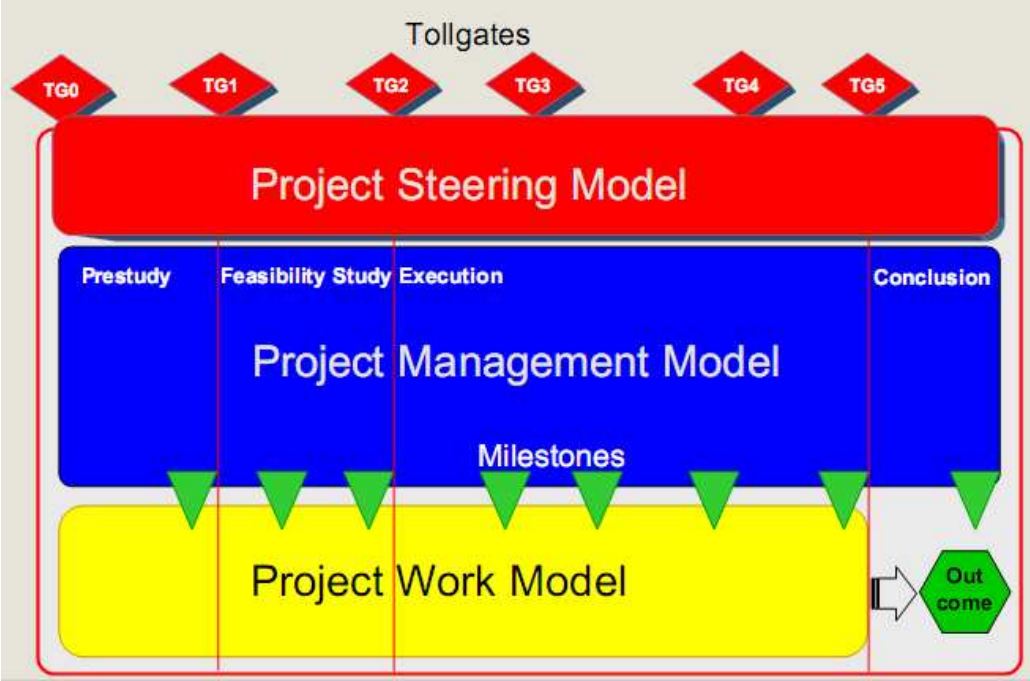
## Toll Gates

All the sub-processes start with a Toll Gate (TG). The Toll Gates are mandatory business decision points that must be passed in order to enter a phase. The formal decision to cancel the project or continue is made by senior managers or by the sponsor. The decision is based on factors like the benefit of the outcome, the use of resources, and the project status in terms of deliverables and the progress. These factors change during a project. For instance a market window can have already passed before a project is completed, leading to a substantial decrease in the benefit of the customer.

A TG can also be a trigger for all processes that are connected.

## Milestones

Milestones are placed in the work model part of PROPS, and are points in the process where decisions taken by the project manager and his team. A milestone is also a collection of criteria and before each milestone, a milestone review is performed to check that these have been met.




## The pre-study phase

The process begins with a pre-study. In the start of the pre-study phase we find Toll Gate 0, which serves as the entry point where the decision to do the pre-study of the project is taken. The purpose of the pre-study itself is to do a requirement analysis. Based on the expressed and unexpressed requirements and needs of the customer, the project feasibility is assessed from technical and commercial viewpoints. This includes making an assessment of what resources, e.g. human- or financial-, are required by the later phases in the process. Depending on the product, a prototype can be made at this stage. In the end it is verified that the financial resources are at hand.

Concluding the pre-study, a report is written. This report is used in transition to the next phase.

## The feasibility phase

As with the Prestudy phase, the Feasibility phase begins with a Toll Gate, the TG1. This again makes for calling business decisions for the project, like continuing, adjusting or cancelling the project based on the results of the previous phase. Before reaching a decision, meetings are held to for the managers to discuss the impact of the project to the business and it's influence

on the current and future projects. As a lot of time and effort has already been invested in the project at this stage, the threshold for cancelling is naturally higher than at previous Toll Gate.

During the phase itself, a plan for the successful execution of the project is made describing participation, what specialists are needed and how teams are structured by defining how different roles in the teams are constituted. At Ericsson, this feasibility phase involves time and budget planning, planning resources, building a business case, making quality plan and risk analysis. A project specification is made as well along with several other organizational planning decisions. The project sponsor may influence the planning as well, as he may set conditions for the financing of the project.

## The execution phase

Just like the other phases, the execution phase starts with a Toll Gate, but differentiates in the way that it has two additional Toll Gates. At the first one, TG2, a meeting is held where document decisions are made and other departments are triggered. For example the marketing department could be instructed to prepare strategies for the new product.

After TG2 the execution of the plans from the feasibility phase is started in order to produce a product. In our case the product is software, and the producing involves software development.

At TG3 the project manager reviews and assesses the ongoing process, and uses this information to make necessary adjustments, if any, to revise the plan. Change handling is also done continuously in a smaller scale project control along the project time line.

The last Toll Gate during the execution phase is TG 4, and here the actual quality of the product is reviewed. This is made in a test report containing a quality assessment regarding number of defects, and the severity of these, discovered during quality control. The Product Manager needs to accept the product before continuing the process which at this phase is to start the hand-over of the project outcome to the internal receiver or external customer.

## The Conclusion phase

The last stage of the process is the conclusion. It begins with Toll Gate 5, which is where the project outcome is accepted and this triggers the mass

production of the product, also called HVM (High Volume Management) in Ericsson.

The conclusion phase ends with an evaluation of the project . It is the closed and a final report is made to document valuable experiences for future use.

## Props Summary

The PROPS model is highly flexible and it does not stipulate either time-consumption or lead-time for a project. It is also flexible to what technologies to use in the underlying work model.

At Ericsson Grimstad, Customer Development Team has a customized process for the Software Development Process.

# 4 State of the art: Analysis of related work

Software development describes the collection of technologies that apply an engineering approach to the construction and support of software products. Technology used involves tools, concepts, techniques, principles, development approaches, methods and even software processes.

In this chapter, we review the state of the art related to software process improvement.

## 4.1 Software Process Improvements

### 4.1.1 Background

Over the last twenty years, a consensus has raised that software process improvement (SPI) is highly important for software companies. A myriad of publications has been published in this field. Hansen et al. has reported (Hansen, 2004) a review of 322 representative contributions to the Software Process Improvement (SPI) literature. Notably, the Software Engineering Institute (SEI) at Carnegie Mellon University has drawn most of the picture of the current shape of SPI.

According to Hansen, (Hansen, 2004) the SPI approaches can be broadly classified into: norm-driven and problem-driven. A similar taxonomy of process improvement activities as either maturity model or improvement method has been adopted by F. Cattaneo (Cattaneo, 1998).

Norm driven approaches (Arent et al., 2000; Arent et al., 2001) are built upon standard model of software process improvement. The main concern is to align to an exiting software process to the prescribed norm. Problem driven approaches (Iversen, 1999) implicate solutions that derive from specific problems. They mainly address problem identification and solving activities.

### 4.1.2 Norm driven approaches

Norm driven approaches to SPI share some common features. They describe how the process should be standardized to reach a certain level of efficiency. They attempt to eliminate the difference between the process to be improved and an existing baseline. Norm driven approaches embraces organizational level, project level, team management level and the individual level. Maturity models are the representative class of norm driven approaches. Maturity models draw the profile of an ideal software process. They prescribe standards for software firms on how individuals, teams, organizations should operate in

order that the organization achieves a higher level of maturity. Maturity models assume that a process is measurable. They provide mechanisms to assess the maturity process using various questionnaire-based techniques. A maturity level is assigned. Subsequently, the assignment is followed by a prescribed roadmap to attain the next maturity level.

Typical examples of maturity models include CMM , SPICE/ISO15504 standard. (ISO/IEC, 1998) and IDEAL (McFeely, 1996).

We should emphasise there is no clear boundary between Maturity models and improvement methods. In fact, the two concepts are fairly orthogonal concepts. In this perspective, F. Cattaneo states: "A maturity model does not necessarily determine a specific improvement method. Similarly, an improvement method may or may not exploit different maturity models" (Cattaneo, 1998).

For the sake of clarity, we cite the most prevalent norm driven approaches in the literature.

## CMM

The Capability Maturity Model (CMM) has gained a lot of attention in software process improvement field. It is likely of the most known and used norm driven approaches. CMM is defined as "a description of stages through which software organizations evolve as they define, implement, measure, control and improve their software process" . The Software CMM is a staged model. Maturity level 5 is a continuous level of improvement entailing innovative technological improvements and incremental improvement. Variants of CMM including other areas of interest has been proposed in the literature such as Software Acquisition CMM, System engineering CMM, Integrated Product Management CMM, and People CMM. The the dimensional expansion of the CMM was a key motivation for SEI to integrate all CMM emergent models into a unified CMM model, called CMM Integrated (CMMI) . The motivation behind CMM seems to be creating a secure theoretical grounding with less emphasises on the practice. It defines how the process should be. It classifies the process efficiency according to theoretical prescription.

## ISO 9000

The ISO 9000 series of international standards for quality management was first published in 1987. ISO 9000 is a general model for quality systems to be used in a wide range of application domains. ISO 9000 focuses on the company ability to control and ensure the quality of the products/services it delivers. It therefore takes into account some aspects related to both internal and external coherence. As a remark, while ISO 9000 take in some consideration the issue of coherence, it lacks the detailed domain knowledge embedded in CMM. This is

quite obvious, since ISO 9000 is supposed to be applicable to any domain (Stelzer et al., 1996).

## SPICE

SPICE (Software Process Improvement and capability improvement) is a framework that includes a set of international standards for software improvement and assessment. (Alec, 1993)

The objective of SPICE is to provide a common approach and framework for assessment and improvement. The so-called Process Improvement guide (Kiston, 1997) describes the steps to perform within a complete cycle based on the SPICE assessment technology. The phase of the analysing the assessment results in a action plan. Basically the approach advocates is emphasising the definitions of goals and the usage of measurement to show quantitatively the current status of processes and practices against a general understanding of software best practices of CMM.

The phases of action plan production in SPICE are broken down into three steps:

-Identify areas for improvement based o the measurement output, the organization's improvements goals, effectiveness measurement if available, risk factors, and any industry norms and benchmarks that provide comparison framework for assessment results.

-For each of these areas, a target for improvement should be defined either in terms of process effectiveness (e.g percentage of project with an accuracy of 10 percent of effort estimates) and/or target process capability profile as defined by SPICE (capability level for given process). This involves defining goals, devising the rights metrics to measure their achievement, and setting appropriate target values.

-Finally, an action plan should be derived covering improvement actions with associated process goals and improvements targets responsibilities, initial estimates of effort, benefits and schedule, and risks to products and to organization if actions are taken or not taken.

## Bootstrap

The Bootstrap method is a framework for assessing software process maturity. Bootstrap combines the following approaches: the Software CMM; ISO 9001/9000-3 etc.

The project involved seven partners from five countries. It developed the bootstrap software development process assessment and improvements method (Grady, 1992). Bootsrap claims to be equally applicable and cost effective for software organizations of any size and in any application domain. That is in

contrast to the CMM, which has a natural or perceived bias (due to its origin) toward large organizations and toward defence and other real time applications.

An essential part of the Bootstrap approach is that assessment lead to improvement actions plans. A means of developing action plans is an integral part of the method.

The primary output of a Bootstrap assessment lead is a maturity profile, which shows the maturity of each of the component parts of the total process. It is, however, possible to convert the maturity profile to the CMM single figure maturity level.

Bootstrap operates on higher degree of granularity than CMM, in fact, it assesses the individual projects rather than the entire organisation. The result of the assessment is a number between 1 and 5 for instance a department could be rated at 3.4. The interpretation of this number is that the department has achieved a better improvement than CMM level 3 but it is does not still attain CMM level 4.

### 4.1.3 Limitations of norm driven approaches

Criticism to norm driven approaches has been addressed in the literature (Grady, 1992).

Norm driven approaches measure one attribute (maturity) of one factor of production (established process). They take no account of other possible process attributes (such as fitness, flexibility, etc ). The weakness comes precisely from a massive simplification of the process to be used that is not accompanied by a balanced understanding of how process interacts with other production factors to impact overall performance measures (Fenton 1991) . Process maturity models measure maturity by counting the presence of absence of standard practices. This is a very simple mean of measurement. One interesting feature is that, by their very nature, such models are unable to measure the maturity of individual practices: they can only provide indicators but not exact metrics. It also can not fairly cope with organizations whose set of practices, for good and deliberate reasons, varies from the standard set. Process models are unconcerned with cause effect relationships (Fenton 1991). They are based on a very simple proposition, that software product quality is a function of software process quality. Such a statement of a static equality can be vastly misleading, because it ignores the real world chains of events which are interposed between process change and a product change, the speed at which they unfold, and the extent to which they are affected by other causes.

Hansen concluded in (Hansen, 2004) that the shape of SPI is mainly dominated by CMM. CMM represents the most famous and deployed norm driven approach in the software firms. Criticism to CMM has been addressed in the literature. CMM focuses on the institutionalization of a standard process neglecting the individual level. The goal of CMM is achieving a higher CMM level, a goal that is not synonym to achieving better software process. CMM

mainly focuses on a uniformization of the software processes. It aims at creating a world wide global process. However, the process is tailed in America may not be suitable for a Scandinavian company. Models do not consider the organization culture and politics. The link is tenuous between CMM and the specific circumstances of the organizations. Bach has showed that applying CMM makes the SPI vague and the enterprise not aware of the problems to be solved, some organizations do not identify the goal that CMM is intended to address (Bach, 1994).

### 4.1.4 Problem driven approaches

Like norm problem driven approaches, problem driven approaches aspire at improving software processes. The singularity of such approaches is the fact that they are more concentrated about recognizing the problems within a software firm and solving them in a more systematic fashion. This is different from norm driven approaches that focus rather on establishing models for software development. In contrast to norm driven approaches that impose burden on the process (model to follow), problem driven approaches supply a higher degree of freedom by envisioning methods to improve the process that are rather specific to the organizations in question. In the latter approaches, assessment is less important than in norm driven approaches.

Remarkably, problem driven approaches are to high extent inspired by the effort deployed in the manufactory field . These approaches presume that the software process and the manufacturing process share common features that render the same improvement methodologies applicable and profitable for both of them.

Norm driven approaches and problem driven approaches agree on the same perception of the software process: a repeatable activity that compromises a set of detailed sub activities and procedures.

A typical example of problem-driven approach is the Japanese software Factory Approach, which is not defined as model but rather as a practice that evolved and created values from the industrial field. For the sake of clarity we cite the example of Toshiba that applied a three phased model in its manufacturing process. Applying these steps in a case study highlighted an upgrade of quality and productivity (Matsumoto, 1987).

Furthermore, The Application of Metrics in Industry (AMI) is a project that aims at combining planning actions with Software process assessment. The application of AMI starts with assessing the project environment and defining the primary goals for improvement. The improvement program is sustained by plan tracking to detect deviations from the project goals or the standards and procedures the project uses.

The Experience Factory is a problem-driven SPI which paramount principle is separating the development organization from the experience factory. The experience factory is a physical organization that process information from the development organization and returns a feedback. The experience factory assists continuously the development process by providing goals and models tailored from previous project experiences.

## 4.2 Software developments models

The software development models can be classified into classical software development models and agile development methodologies. We provide here an overview of these two trends.

### 4.2.1 Classical Software development Models

Programming methodologies can be perceived as mainly oriented to the conceptual principles of software engineering. A set of more programmatic technologies developed in software engineering is known as the software development models, such as the waterfall (Royce, 1970), prototype (Boehm et al., 1984; Curtis et al., 1987), spiral (Boehm, 1988; Boehm and Bose, 1994), V (GMOD, 1992), evolutionary ( Lehman, 1985; Gilb, 1988;Gustavsson, 1989) , and incremental (Parnas, 1979; Mills et al., 1987 ) models.

Supplementary to the above development models, a variety of detailed methods have been proposed for each phase of the development models. For instance, just for the software design phase, a number of design methods have been in existence (McDermid, 1991], typically flowcharts, data flow diagrams, Nassi-Shneiderman charts, Program Description Languages (PDLs), entity-relationship diagrams, Yourdon methods, and Jackson system development. Of course, some of these methods may cover multiple phases in software development.

These models can be classified as classical models that has been adopted and designed some decades ago. They attempt to provide a set of guidelines for the design and implementation of software at system and module levels. However, these approaches have been focused on technical aspects of software development lifecycles. Organizational and managerial methodologies and processes have not been covered. Detailed descriptions and applications of existing software development models may be referred to the classic software engineering books (McDermid, 1991; Pressman, 1992; Sommerville, 1996).

### 4.2.2 Agile Development methodologies

A major departure from the traditional software development model has been marked with the advent of agile development methodologies few years ago (Dybå et al., 2008).

In a recently published survey about agile methodologies, Dybå describes the agile development methodologies as:

*"Methods for agile software development constitute a set of practices for software development that have been created by experienced practitioners. These methods can be seen as a reaction to plan-based or traditional methods, which emphasize a rationalized, engineering based approach in which it is claimed that problems are fully specifiable and that optimal and predictable solutions exist for every problem. The "traditionalists" are said to advocate extensive planning, codified processes, and rigorous reuse to make development an efficient and predictable activity."* (Dybå et al., 2008)

The agile development is a merging development methodology that is mainly the contribution of practitioners who defined most of its current shape. In 2001, the "agile manifesto" published the four core underlying values of agile (Dybå et al., 2008)

Value 1: Individuals and interactions over processes and tools.
Value 2: Working software over comprehensive documentation.
Value 3: Customer collaboration over contract negotiation.
Value 4: Responding to change over following a plan

Based on these four core values defined by the manifesto, variant of the agile methodology has been proposed. For the sake of brevity, we cite the most prevalent of these methodologies while putting emphasize on Scrum since it will intend to use it in the section solutions.

#### Dynamic software development method (DSDM)

Like other Agile methods, DSDM assumes an iterative life cycle. The DSDM process compromises three phase:

The "Functional Model Iteration" phase, the "Design and Build" phase and the "Implementation" phase. The intent of DSDM is for each project to define how the iterating will be done so that the needs of the project are met. The three iterative phases would generally be turning concurrently, and feedback from "Implementation" and "Design and Build Iteration" to the other phases could happen during any iteration. The DSDM process provides a structured

set of activities with feed-forward and feedback loops, but it allows a large degree of freedom for any particular project to define exactly how those activities are assembled to define the project's life cycle. (Dybå et al., 2008)

**Extreme Programming (XP):**

Extreme Programming (XP) is probably the most widely recognized of the Agile methods. XP focuses on 12 practices are its defining features. The principles are: The Planning Game. Small releases, metaphor, simple design, test first, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, one site customer, coding standards.

**Lean development (LD):**

Lean Software Development is not a software development method. Rather, it is a set of principles and tools that an organization can employ in making its software development projects leaner. The principles behind LD are drawn from the world of lean manufacturing, namely the Toyota Production System, and although some LD tools relate directly to lean manufacturing principles, many do not. LD is characterized by seven lean principles that are elaborated into 22 Lean Software Development tools. For a comprehensive review of Lead Development we refer the reader to an outstanding book about Lean written (Poppendieck et al., 2003)

### Scrum

Scrum is an agile development method. Scrum as it exists today grew from its beginnings in Japan in the mid-1980s (Takeuchi; 1986). Takeuchi (Takeuchi; 1986) concluded that collapsing phases of product incarnated in Scrum Sprints yields a higher productivity and shorter timeline.

Each increment of a Scrum project is developed in a "Sprint." A Sprint is a time-boxed development increment that is generally set at 15 or 30 days in length. The Sprint is characterized by its goal and a set of functionality that it is expected to deliver.

### Scrum practices

As with other Agile methods, Scrum is defined not so much by its process as by the practices that comprise it. We describe here the main practices of Scrum.

## Product Backlog

The product backlog is at the heart of Scrum. This is where it all starts.

*"Product Backlog is an evolving, prioritized queue of business and technical functionality that needs to be developed into a system"*

(Book Agile Software Development with Scrum , page 32)

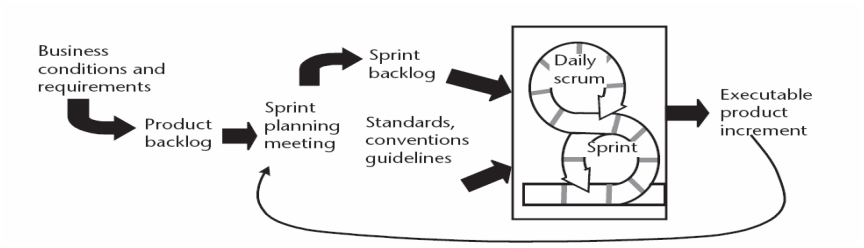
.

The Product Backlog is the sum total of the work that remains on the project and includes everything from major features to bug fixes. Any stakeholder in the project can contribute to the Product Backlog at any time, but it is the Product Owner who has the primary responsibility for determining the priority of backlog items. The primary measure of progress in a Scrum project is the change in the number of items in the Product Backlog over time. It may grow in early Sprints as stakeholders gain an understanding of the system being built, but ultimately, a pattern of steady decrease in the size of the Product Backlog is expected. If this does not materialize, or if it is not fast enough, then hard decisions must be made about the project's scope.

## Sprint Review

Each Sprint ends with a Sprint Review meeting in which all stakeholders come together to review what was developed during the Sprint. This review includes the entire development team, the customer, and management, and it allows each person to learn from what was developed during the Sprint and to prepare for the planning session for the next one.

The Scrum process, shown in figure below, is incremental.




### The Scrum Master

"The Scrum Master is responsible for the success of Scrum". (Book Agile Software Development with Scrum , page 31)

Although Scrum defines this as a new role, in traditional projects its responsibilities are often taken on by an existing position such as project manager or team leader. The primary responsibilities of the Scrum Master are to:

☐ Ensure that the Scrum practices are followed and that the values behind Scrum drive enactment of the process.

☐ Be the interface point among management, the customer, and the Scrum team. Of primary importance are:

☐ Communicating project status;

☐ Removing impediments to progress

## Scrum Teams

"A team commits to achieving a Sprint goal. The team is accorded full authority to do whatever it decides is necessary to achieve the goal2 (Book Agile Software Development with Scrum , page 35)

. Almost all software development involves teams. The key difference with Scrum is that the team freely commits to what they believe they can produce during each Sprint, and they are empowered to make whatever decisions they must to fulfil those commitments.

They may even change the details of the functionality to be delivered as long as they believe they will still achieve the Sprint goal. If they become convinced that the Sprint goal is beyond reach, they are empowered to abort the Sprint, which would immediately result in a new Sprint Planning session. By doing so, they will force the other project stakeholders to reassess the new information they have learned so they can all work together to set a new, achievable Sprint goal to get the project back on track.

During the Sprint, the team self-organizes and self-directs, and their authority even extends to being able to:

☐ Change the functionality to be delivered by the Sprint as long as the Sprint Goal is still achieved.

☐ Abort the Sprint if new information leads them to believe its Goal or Backlog is no longer achievable or relevant.

Assuming the team does not abort the Sprint, it ends with the delivery of the promised executable product increment.

As with the other Agile methods, Scrum's time-boxed increments provide a mechanism for all project stakeholders to learn about the system being built on a regular basis. In the case of Scrum, this happens every sprint end.

This level of autonomy is foreign to most organizations.

## Daily Scrum Meetings

"The Daily Scrum meeting is where the team comes to communicate" (Book Agile Software Development with Scrum , page 32)

. The Daily Scrum (the defining feature of Scrum) is a short 15-minute meeting that takes place every working day. It is the forum where team members exchange information and others may come to listen□but not speak. To keep the meeting short, all deliberation and discussion is relegated to meetings of interested people after the Daily Scrum. During the Daily Scrum, each team member answers three questions:

□ What have you done since the last Scrum?

□ What will you do between now and the next Scrum?

□ What got in your way of doing work?

The third question provides the Scrum Master with the information he or she needs to be effective in removing impediments to progress and ensuring the team continues to be productive.

## Sprint Planning Meeting

"Customers, users, management, the Product Owner, and the Scrum Team determine the next Sprint goal and functionality at the Sprint Planning meeting. The team then devises the individual tasks that must be performed to build the product increment" (Book Agile Software Development with Scrum , page 47)

Each Sprint begins with this planning meeting. The critical outputs of this meeting are:

□ Sprint Goal - The objective that is to be achieved during this Sprint.

□ Sprint Backlog - The subset of the Product Backlog that will be completed during the Sprint.

The Product Owner is the sole arbiter of the priority of the Product Backlog items. But only the Scrum Team can commit themselves to completing specific work. As a conclusion of the sprint meeting stakeholders will have agreed to a Sprint Goal and Sprint Backlog to which the Team is willing to commit.

## Sprint Review

"The Sprint Review meeting is a four-hour informational meeting. During this meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint".
(Book Agile Software Development with Scrum , page 54)

This meeting provides a concrete picture of the progress achieved during the Sprint and lays the foundation for the next Sprint Planning meeting.

# 5 Software change management problems at Ericsson

This chapter presents change management problems derived from the case study and the literature review. As master students we have a fresh perspective. Thus, we are able to objectively question steps in the process to ameliorate it.

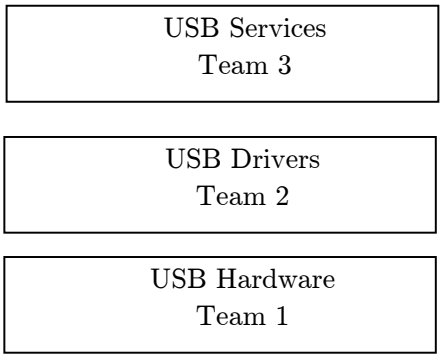
## 5.1 Connection between sub CRs

An important number of customer change requests usually involves the coordination of different departments. Whenever a CR affects several technological parts of the product, it is divided into cross functional sub CRs. This kind of change requests are called cross functional request because the deployment affects many technological aspects that can not be done by one team. In this case, the CR is divided into complementary sub CRs each within a field of specialization.

EMP is organized in specialized teams such as WLAN, USB and AT-commands. When a cross functional CR is received, its sub-CRs are assigned to these teams according to their specializations.

To ensure that the premises to complete all CRs is present and available; EMP management rely that a mutual coordination and exchange of information between teams is established.

An example of a cross functional CR that may affect several teams is a new feature called “Charging by USB”. Such a change requires that the USB port is used as source of power to charge the batteries of the mobile device. Developing and maintaining this feature is a project that involves several distributed teams. One team is responsible for the hardware USB-block, another for the low-




level drivers and a third for higher level USB services. The USB driver developers must coordinate the interface on a bit-specific level with the hardware designers. The hardware designers must also document and inform the team responsible for USB services on how to operate the USB hardware. To ensure maximum synergy between teams, this process requires an efficient approach for coordination and exchange of information.

Nevertheless, this is not always the case at EMP. In our in-depth study of the software development at EMP customer organization we identified sub CR related handling problems. The interviews conducted show that many sub-CRs are not as well aligned as they should be. The interviewees singled out that there exists no mechanism at Ericsson to ensure coordination between two separate teams working on CRs. Some co-work tasks are invisible to other process flows. Consequently, distributed teams find themselves waiting for an input from each other rendering the project in a blocked state.
To escape such a blocked state and to continue progress, teams waste a significant effort and time on an overwhelming amount of informal communication like email and phone calls.

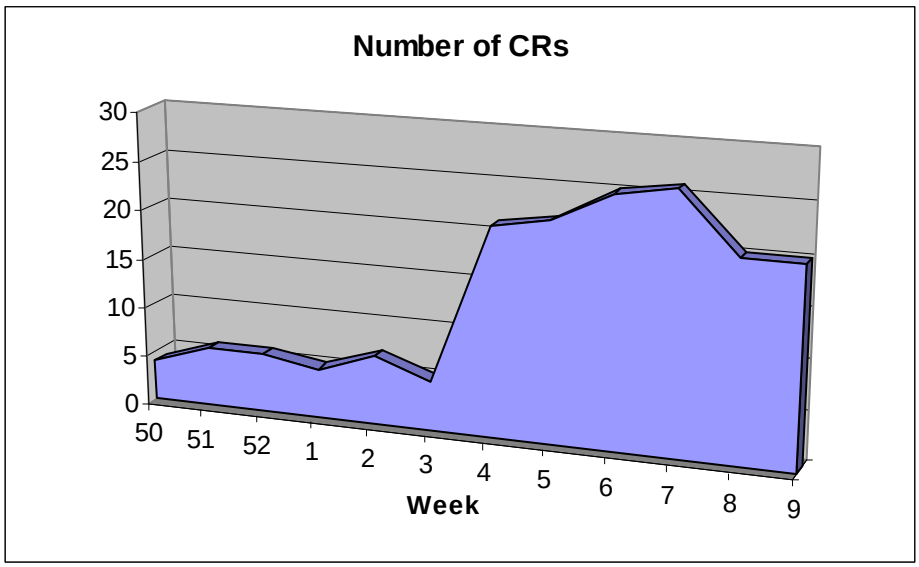
Therefore, it is clear that coordination between two separate teams working on complementary sub-CRs is not supported enough by the process. The coordination tends to rely on personal contacts.
An example that illustrates this at EMP is CRs involving Network signalling and AT commands. Both sub-CR processes are performed separately; however, tight coordination is needed between them. According to an engineer interviewed from the AT commands teams, the activities where collaboration is substantial are not clearly defined in the process leading to time delay.

Moreover, more problems due to this shortage of coordination usually surfaces during the integration phase. In fact, one of the interview developers reported "the hardware and the software process are too mangled and separated such it is difficult to have something working during the integration phase.

## 5.2 Handling CR

In an interview conducted with the head of the customer development organization at EMP Grimstad, he confirmed that the Customer Development Organization is struggling to handle CRs as soon as they arrive. This problem does not stem from capacity scarcity. However, it is due to non homogeneous distribution of the work that makes the resources unavailable during peak time.

In fact, CRs do not arrive like "marching soldiers" at regular intervals. The inter-arrival time between two subsequent CRs may vary considerably. Quick successions of CRs lead to peak working periods and high process overhead. By contrast, slow succession of CRs results in idle working periods. The figure below depicts the distribution of the arrival of CRs during a period of 12 weeks from week 50/2007 to week 9/ 2008 at EMP Grimstad.




The figure confirms that the distribution of CR arrivals is not uniform over a period of three months.
There was an important increase of the number of CRs starting from week 4 in 2008. This increase has resulted in an unsustainable workload at the company. The congestion of CRs are the main cause of long lead time since a large part has to be queued.
In this sense, peak times pose key challenges to the EMP customer organizations. That why we are committed to design a solution that maintains a steady and sustainable level of effort.

## 5.3 Scheduling

The considerable uncertainty inherent in software development as well as the diverse nature it comprises, makes scheduling essential when planning and managing a software project. However, most software developers do not separate important requirements from the less important ones. As a consequence, it can be problematic to achieve the best possible software system.

In order to achieve this, a manager has to take into consideration different aspects of a project to be able to make good scheduling. For instance, the manager must consider the degree of specialization of the teams or the strength of the coupling between the components. A strong coupling between the components results in a higher probability of a design change propagating throughout the software. This may produce a lot of rework within the different components. A high degree of specialization on the other hand, means that a team will be more productive on a certain set of components, and less on others.

These are the circumstances that make project scheduling a difficult challenge for software project managers.

During interviews we discovered that EMP Grimstad uses no tools for assisting scheduling. Currently, there are two strategies for scheduling customer requirements at Ericsson Grimstad. One is the "first come, first served", where all customer requirements within the different levels of priority, set by the CR owner, are processed as soon as possible with the assumption that they should be completed as soon as possible. In reality this is not necessarily what the customer really needs or wants.

The other strategy is to give total commitment to one project for short period of time, and allocate all available resources to completing the change requests assigned to this project.

CR's are given priority by the employee registering it or the by the customer. Priorities range from

| | |
|---|---|
| *low | - can wait |
| *medium | - scheduled to next release |
| *critical | - immediate response (system failure) |

This quantification is too coarse for making good scheduling decisions.

A substantial improvement to the Software Development process can be achieved if the managers have guidelines to help schedule their projects in the most optimal way, despite the many uncertainties that exist in the area of software industry.

## 5.4 Errors

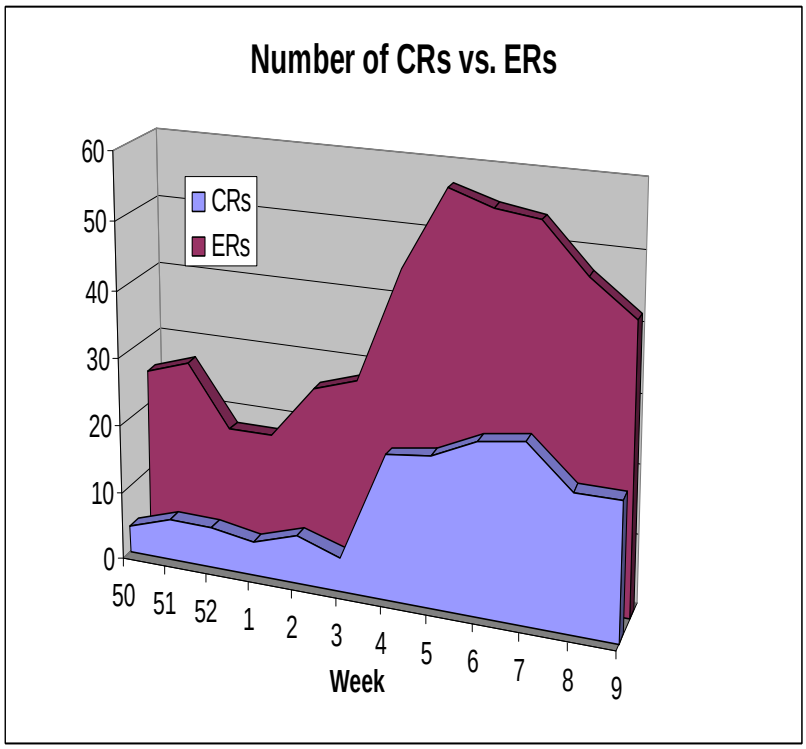
Errors are a recurring problem in software development. Many studies have confirmed that errors have a negative impact on quality and productivity (Westland, 2002). Errors can also become very costly. A specific study performed at Ericsson Telecom AB showed that within Ericsson, the correction of one fault found in service amounted to $7,000 on average (Ohlson et al., 1996). When discovered at a late stage, errors are even more expensive as it is generally accepted that the cost of errors raise exponentially with each new development phase. The reason of such a high cost is that errors heavily influence time to market as it disrupts the stability of the software process. As effort is turned into correcting the errors instead of developing the product, the overall productivity is decreased. Consequently, keeping the number of errors down to a minimum is an important factor to the cost and schedule of a project (Jones, 1994).

To achieve this goal, mature companies implement standard procedures to analyse these error reports and then use the feedback of this analysis to avoid future errors. One such mechanism is used in the highest level of the Capability Maturity Model, and is called Defect Prevention. This mechanism has proved its efficiency in many organizations.

Unfortunately, this is not modus operandi at EMP Grimstad. In fact, there is no resources dedicated at Ericsson Grimstad to analyse collected errors to draw leverage from previous projects. Error reports are stored in an incident database after merely classifying them by their priority, status (e.g. investigation, analysed, accepted/rejected, ongoing/pending, and completed) and in which software module the error was discovered.

Another problem we identified by analysing data collection (CPO CR-team statistics) at EMP Grimstad, is that the number of error reports (ERs) is relatively high compared to the number of change requests (CRs). This suggests a condition of reduced productivity due to valuable resources being lost to the correction of ERs. In short, the fewer errors need to be fixed, the less of the software needs to be rewritten.

The figure below depicts the number of CRs and ERs that were under investigation during a period of 12 weeks from week 50/2007 to week 9/2008 at EMP Grimstad.




During the sampling period, the ERs outnumber the CRs by an average factor of 3,34. It is evident that if we attack the cause of this relationship, thereby reducing the factor, it will unquestionably assist Ericsson Grimstad in delivering a product of high quality at the right time.

## 5.5 Process complexity

The complexity of the software process has been the object of scrutiny of a number of notable studies(Dandekar, 1996; Dandekar, 1997; Dewayne, 1994). Dandekar highlights the complexity of software process as an important driver of lead time. In the study (Dandekar, 1996), the author argues that a main cause of process complexity stems from the "non-added customer" focus tasks. A "non-added customer" focus task is defined as activities that are of no interest to the customer such as unnecessary rework due to design weaknesses and extensive documentation. Such "non-added" tasks can be identified by revealing the composition of a complex process using a visual presentation of

the process. Formalized visualization tools is an efficient method to represent a process at various levels and understand its architecture (Carr, 1995).

Subsequently, to gain an understanding of the software CR process at EMP, we relied on a visual presentation which is available through Ericsson Internal documentation. This visual presentation has captivated our attention as it is remarkably complex. Based on this presentation we identified the documentation process subsystem as a major component. To investigate this finding, we interviewed developers involved in the documentation process.

The interviews revealed a consensus that a lot of the produced documents are obsolete, time consuming, and without added value. For the sake of clarification, we cite some of the proclamations of the interviewees:

-"Adding two lines to the code takes one week of documentation"

-"In the design document, the text is too much, it would be better if we can have more graphics and less text."

Besides an extensive documentation process, it was expressed that code review, with the aim of aligning the code to the Ericsson standard style, mostly is considered to be an unnecessary step that consumes considerable time. Some experienced engineers were complaining about code review, claiming that it is of no need and/or that it is cumbersome. One of the developers said: "In code review, you have to change variables names that are not in conformity with Ericsson standards. This takes time and is usually done in iterations. You must go through the code again to be sure that this change does not introduce any errors."

In addition, we identified that looping structures in the document process usually results in extra or duplicated work, thereby wasting valuable time. As an example of inner loops, we have identified document review. After documenting a section of code or design, the author sends the document to the reviewer. The reviewer checks the quality of the document and issues remarks to the author in order for him to upgrade and/or correct the document. The review is repeated a number of times until the document is correct.

## 5.6 Testing challenges

Significant efforts are spent by software development companies to test their products in order to assure their products are of a high quality level. Software applications are generally getting more and more complex, and this

increases the chances of errors. It is not uncommon that about 50 percent of the total development time is dedicated to testing and fault removal (Harrold, 2000)

Similar to the standard waterfall development process, EMP begins development with a design phase that creates a set of final programming specifications. Then developers start coding the product before the test department begin working on its test case development. The test department waits until development is finished with coding before starting working on test case development.

However, this creates a twofold problem. First, with increased pressure to get the product more quickly, testing is rushed. In parallell there is a demand for increased quality which puts even more pressure on the test department..This creates a strain that can not be handled by the existing process.In this perspective, a process improvement that can identify errors on a early stage is much needed.

Second, according to the inherent nature Waterfall model, there is a time gap between the end of the development of one CR and the results of the tests. At EMP, the interview confirmed that the results of the test department are too late, the error reports are issued in a time when they moved to other projects. Therefore, the EMP engineers has to go to mental settings again to remember the erroneous CRs that they have developed 2 months ago.

"Therefore the engineers may have to go through mental settings and become less productive if spread between doing a current CR and handling Errors reports of a CR that I have done three months ago."

A part from the problems that raise from the characteristics of the waterfall model, our study have identified that the process lead time is dominated by the testing activity referred as the bottleneck activity. In fact, when it comes to testing CRs, congestion is one of the main drivers of lead time at EMP. In fact, only two engineers are dedicated to test CRs. The remaining testing resources, almost 33 engineers, are exhausted by software release tests.

However this situation is controversial. In fact, during an interview with the manager of the customer development team EMP Grimstad he affirmed that half of the new functionalities introduced into the platform come from change requests.

Effort in classifying and quantifying the relationship between faults and failures can prove to be a valuable investment to aid the development of module tests and the system verification department in pinpointing which phases and what activities should be improved to get more efficient test strategies.

## 5.7 Long build time

Excessively long build times is an obvious bottleneck in software development and is perceived as frustrating and demoralizing by the developers because of the delay in the speed of which they receive feedback, leaving the developers in a state of uncertainty. The ability to have shorter build time is not only valuable for the morale of the developers. Having feedback on whether changes are successfully integrated into to the system within a reasonable time, creates an efficient feedback mechanism that increases productivity and quality in contrast to discovering a small, careless mistake and having to run the build all over again. Moreover, long build time can have the consequence that developers take short-cuts by not running a build locally before checking in minor changes they feel confident in will not create any errors. As well developers may skip running tests.

Shortening the build will make software development teams able to minimize cost of integration. It can give a team both confidence and speed especially when implementing small incremental changes. Thus, the team members will be able to spend more resources on other phases of the project, or continue to the next project.

The build time at Ericsson can be classified as excessively long. Due to a complex and huge code base, a full build can use 4 hours to complete using the workstation of a developer. Usually these builds are started at the end of the day and therefore go under the name “night build”. The partial builds take much less time to complete. Depending on the degree of changes made, a typical partial build can take about 20 minutes. This is still too much, considering the above mentioned disadvantages.

There are many things that the developers can do while waiting for the build to complete. They can for instance work on documentation or perform routine tasks like checking email. However, developing quality code is a thought intensive process and changing area of focus is counter productive. Because pulling focus from coding will distract the developer, it requires a considerable period of time before he is back to his previous level of productivity.

## 5.8 Communication problems

Communication plays a decisive role in the efficiency and success of an organization.

EMP is additionally a distributed organization where collaboration involves local and cross site communications. Such an environment raises a need for informal, distant communications for the cross site communication case, and face to face communications for the case of the local site (Ahuja, 1998). Empirical studies have shown (Curtis 1998; Kraut, 1995; Hersleb 1995) that a shortage of informal "ad hoc" communications among software engineers has a negative impact on a project's advancement.

In Ericsson context, the interviewees highlighted that they are facing problems to work cohesively due to communication problems. They expressed that it is difficult to find the right person when you need relevant information. This communication problem arises between different technological groups when CRs affect several technological parts of the product. The problem is founded in a need of knowledge that is not shared nor documented (Petti; 2002). Finding the right person is an increasingly tedious with regards to the proliferation of the EMP organizations during these last years.

Unlike (Palmer, 1998), we have not received any report indicating trust or conflict management problems. On the contrary, the interviewees confirmed the existence of mutual trust between distributed teams. However, the interviews underlined that remote meetings, based on phone conferences, are not always efficient. They confirmed that the phone conferences do not realise the desired level of mutual understanding. One of the interviewees stated that "The meetings with distributed teams usually fail to meet their fixed goal. Therefore they are rescheduled, leading to delay and stress". Such limitations, this leads to a situation where developers are required to travel to Lund.

We later experienced this first-hand when we were present at a telephone conference meeting with a remote development site in Lund, Sweden. The two sites were both working on components used by the mobile platform, therefore requiring a lot of coordination. The meeting was conducted by a group of employees from Grimstad who in advance had scheduled a voice conference with colleagues from the research and development departments in Lund. Prior to the meeting, microphones and loudspeakers were placed on the conference table.

We noticed that communication during telephone conference was noticeably inhibited compared to real-life meetings as it is missing the visual part that is so often used in non-virtual situations. Additionally, the setting can easily give a notion of a strained, artificial atmosphere as each side is consciously unaware of all non-audible actions at the other location.

# 6 Generic change management process model Recommendations

In the previous section we present identified problems that drive the lead time at EMP Customer Organization Grimstad. To address these problems, we proceed with problem-driven approach. This approach allowed us to target specific problems within the process. An alternative solution was the norm driven approach, which is a strategy to adopt best-practice approaches presuming this will alleviate any problems. We did not use this as it implies that specific problems are never pin-pointed.

The respective solutions are mapped to the problems by the usage of the following matrix

.

| | Prob 1 | Prob 2 | Prob 3 | Prob 4 | Prob 5 | Prob 6 | Prob 7 | Prob 8 |
|---|---|---|---|---|---|---|---|---|
| Sol 1 | | ✓ | ✓ | | | | | |
| Sol 2 | | ✓ | ✓ | | ✓ | ✓ | | ✓ |
| Sol 3 | ✓ | | ✓ | | | | | |
| Sol 4 | ✓ | | | | | | | ✓ |
| Sol 5 | | ✓ | | | ✓ | | | |
| Sol 6 | | ✓ | ✓ | | | | | |
| Sol 7 | | | ✓ | | | | | |
| Sol 8 | | ✓ | | | | ✓ | | |
| Sol 9 | | | | ✓ | | | | |
| Sol 10 | | | | ✓ | | | | |
| Sol 11 | | | | ✓ | | | ✓ | |
| Sol 12 | | | | ✓ | | ✓ | | |
| Sol 13 | ✓ | ✓ | | | | | | ✓ |

## 6.1 Batching

Since EMP produces customized products, many changes addressing the design of the product are usually initiated by the customers. Fewer other changes are initiated by the organization itself. These changes can come on the stage at any phase of the product development posing serious challenges to EMP that aspires to handle them on time and accurately. We refer to section problem to highlight that changes are overwhelming during peak times.

Despite that the CR management mechanisms are highly important with regards to reducing lead time, they have not been the centre of growing interest in the literature. Mandal (Mandal, 1997) reviews trends in this field of research.

According to Mandal many studies have focused on computer based tools for analyzing CRs problems. Other studies dealt with methods of controlling CRs through manufacturing and lessening the effect of product function. These studies stem from case studies performed in companies. Watts (Watts, 1984) has successfully reduced the time delay in a computer products manufactory by reducing the time spent on paperwork. In (Balcerak, 1992) Balcerak performed a valuable study in the field by proposing a scheme for classifying CRs according to their impact of change and the grade of urgency.

We should single out that there are two possible mechanisms for change request handling:

-Processing as soon as they are received
-Batching CRs; a number of CRs are accumulated and processing starts either when the batch reaches a certain size or at regular intervals.

The second technique has proved its efficiency in the case of automotive manufactory (Terwiesch, 1998). In fact, in the later case many changes can be performed in one setup - avoiding disassembly and setup costs. In the same perspective, a recently published pioneering research (Bhuiyan, 2006) has come to the result that batching CRs is shown to be superior to processing them immediately.

The state of practice at Ericsson is to handle CRs as soon as they arrive. According to the findings of (Bhuiyan, 2006) and (Terwiesch, 1998) this is not the optimal way to process CRs.

Based on these results, instead of processing the CR as they soon as they arrive, we propose to introduce batching in order to fasten the development speed at EMP Customer Project Organization (CPO). CRs can be grouped and processed in a batch. This technique yields better results than performing them individually. The main reason for this difference is the lower process overhead when CRs are processed as a group. The amount of work necessary to do the changes is much different if done individually or as a group, the amount of time spent doing communication, design review meetings, and management is much less when CRs are done as a group. Thus, the time consuming activities are done in batch way. It is difficult to determine the intervals of batch without trying this technique in practice. The results of the studies show that the solution is particularly promising. Nevertheless, assessing the solution in an experimental setting requires a large scale experiment involving the cooperation of the entire development team for a period of time exceeding our timeframe. Thus, we can not draw any priori results without enacting it in practice.

Therefore, we urge the practitioners at Ericsson to try this approach.

## 6.2 Adopting agile development: Scrum

During an interview with one of the senior developers at EMP, he expressed that:

"Sometimes, we develop something that the customer did not want. This is because the requirements are not clear for us neither for the customer. This fact is related to the rapidly changing mobile market. The development of CRs should be iterative and incremental, we should develop a little and ask the customer is this what you need. This methodology could support better the changes." Without knowing it, the developer was promoting for the agile new development methodology.

The CR process at Ericsson is based on Props, a process model derived form the Waterfall model. The Waterfall model is a legacy model that dates to 30 years ago. Many literature studies have shown that the waterfall model does not cope with the current software state of practice. Indeed, Larman (C. Larman; 2003) has highlighted statistical evidences of the failures of the waterfall model in the software field.

Moreover, there are compelling evidence that the classical waterfall model does not yield good results. Carolyn Wong (C. Wong; 1984) states that:

"The [waterfall] model was adopted because software development was guided by DoD(US Department of Defense) standards. In reality, software development is a complex, continuous, iterative, and repetitive process. The [waterfall model] does not reflect this complexity."

By the late 1980s, a study conducted at DoD to assess the waterfall practice of development based on the published standard DoD STD 2167 of the waterfall process reported that:

"Of a total $37 billion for the sample set, 75% of the projects failed or were never used, and only 2% were used without extensive modification." (S. Jarzombek; 1999).

Therefore, the report concluded that the waterfall model exhibited a drastic failure over many large projects at DoD. To resolve this critical situation, the Defense Science Board Task Force on Military Software chaired by Frederick Brooks, a known expert international expert in the field released a report that urges DoD to abandon the waterfall model and substitute it with incremental and iterative models. The report highlighted that:

"DOD STD 2167 likewise needs a radical overhaul to reflect modern best practice. Evolutionary development is best technically, it saves time and money"

Therefore, the shortcoming of the waterfall model empirically confirmed motivates us to propose to use an agile development method at EMP as a substitute of the classical waterfall model. This proposal is supported by many performed studies that have confirmed the eminence of agile productivity compared to other traditional development methodologies.

In fact, analyses of an agile methodology implementation (Ilieva et al., 2004) found that a team using agile development methodology outperforms another team using a traditional development by a 42% increased productivity model working when working on a similar project.

Similarly, Layman et al. (Layman et al., 2004) has compared the productivity of a team over two subsequent software releases. In the first release, the team used a traditional development method, in the second release the team used an agile development method. The results were conclusive and showed that the productivity increased remarkably by 46% in the second release.

The case study by Dalcher et al. (Dalcher et al., 2005) compared the productivity of fifteen software teams developing similar projects practicing four different software development methodologies (waterfall, incremental, evolutionary, and XP). The highest productivity difference was reported between the teams using waterfall model and the teams using agile methodology. The results showed that the productivity of agile teams outperforms the productivity of the waterfall model teams by a huge factor l337 %. These results are really conclusive and showed the power of agile development when deployed in small teams.

As an agile approach, we have chosen Scrum as it is a methodology that allows close interaction with the customers as well as being very flexible and open for customization (Mann et al., 2005). The case study of Scrum introduction showed an improvement in customer communication. These are attributes that conform well to the way Ericsson organization handles CRs that involves a continuous customer interaction.

Moreover, in the case study performed by Mann (Mann et al., 2005) the adoption of scrum has reduced significantly the overtime. The developers were very satisfied with the results of the application of Scrum and committed to continue using it in their future projects.

At the beginning of this section, we cited an extract by an Ericsson employee regarding unclear requirements. This claim was further confirmed by another developer in the same team that reported that :"we do not know until the final delivery if this is what the customer wanted". In accordance with these statements, the developers in (Mann et al., 2005) confirmed the same

perception after adopting Scrum by stating that “the Scrum process is giving me confidence that we are developing the software that the customer wants”.

By using Scrum, EMP could embrace a more up to date development model that is able to face the increasing competitivity in the telecommunication market. Trends to adopt the agile development have been already reported in many telecommunication companies such as Nokia. In a similar context to EMP, Nokia has used agile to fasten the development of mobile phones. Agile was proved to boost the productivity if small team and its ability to support change in software requirements. (Vanhanen et al., 2003)

## 6.3 Time boxing

The Scrum Agile development process is a software development methodology that aims at achieving short lead times. Scrum, is an example of tight process which maintains a high pace of productivity through successive delivery in short time boxes. The beauty of Sprint is that it makes use of Time boxing to increase the productivity of the team by imposing fine short time deadlines.

Schuh, in his book (Schuh, 2004) refers to time boxing as:

“Time boxing is a tool that, like any other, can be used for both good and bad purposes.... Agile projects perform time boxing at the release and the iteration levels, so a single agile project contains multiple time boxes.... Time boxes force the customer to make decisions on the short-term direction of the project. Second, they always provide a near-term goal, which can keep the entire team from wandering off target....Finally, time boxes ensure that the team delivers something useful within a short and defined period.” (p. 154-155)

At Ericsson’s Customer Development Organization Time-boxing can be utilized as an interesting concept to turns the development priorities around ( see section priorities). When receiving a CR, a functionality goal should be set and then work is therefore kept until achievement within the duration of the corresponding sprint. If the team is late, it should work overtime and get the functionality goal achieved as close as possible to the scheduled end date of the CR. By keeping the time frame for each increment very short, the managers at EMP Customer Organization help their teams to maintain a reasonable level of pressure almost continually.

With time-boxing, the schedule is immovable. If things do not go as planned, then the functionality is what changes. Redefining the extent to which the functionality must work in this increment. There are many options for "slipping" a function, but the schedule does not slip.

We consider the adoption of Scrum at EMP as an innovative approach. Instead of waiting for the whole CR development to be finished, the manager can track the advancement of the development through increments. This gives a better control of the progress of the CR, and keeps the productivity of the teams at an optimal level.

## 6.4 Introducing connections between CRs

In the section problems, we emphasized the lack coordination between cross functional CRs as one of the main drivers of long CR lead times. In fact, some sub-CRs are tightly related and need coordination along with mutual exchange of information. There is no mechanism at Ericsson to ensure coordination between two separate teams working on complementary sub-CRs (rather than mails and vocal communication).

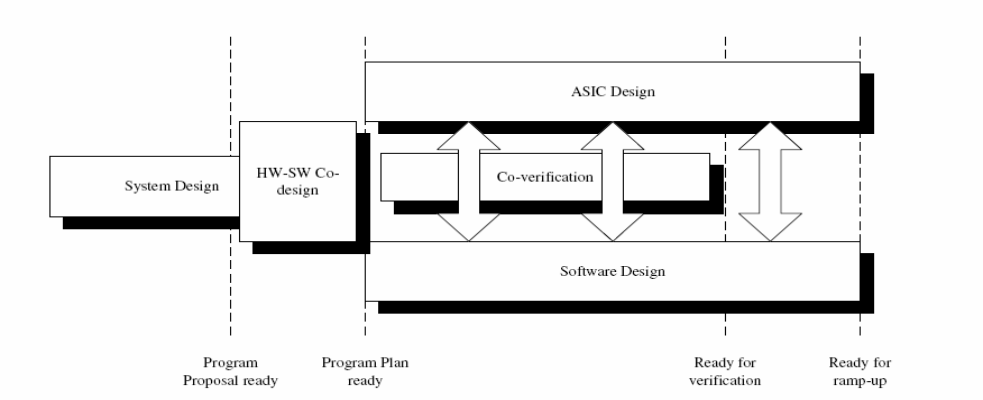
Despite the importance of the later problem, a small body of the literature has addressed this problem. To the best of our knowledge, the only work that was invested in studying the coordination between complementary processes was reported by Jorma Taramaa in (Taramaa et al., 1998; Ronkainen and Jorma 2002; Taramaa Virpi Taipale and Taramaa, 2005). These publications are a part of a research lead at Nokia that have established the grounding of tight coordination between the software and hardware sub processes. They were found to be useful for aligning the software and hardware process at Nokia. Qualitative experimental results have shown the eminence of the proposed approach.

A solution that is susceptible of ameliorating handling interdependencies across the hardware and software process at Nokia can be utilized to handle interdependencies across complementary sub CRs at Ericsson. Therefore we propose to introduce new checkpoints where the sub CRs can be aligned and the results are made available to all stakeholders. The paramount objective is to modify the current process flow for handling complementary sub CR at Ericsson in such a way that it supports cooperative activities.

At these added checkpoints a mutual exchange of documentation is performed. Such a mutual exchange of document fuels the co work and ensures maximum synergy. Each team provides documentation to the other team who need it. In this sense, the output document delivered by one team will serve as a useful input for the other team and vice versa. Delivery of increments of the code and/or design documents can be envisaged at these checkpoints.

Therefore, these deliveries are vital in preventing blocked states referred in the problems section.

Thus, the visibility of the tasks is improved creating a vital linkage between the teams working on the same CR. The checkpoints serve as project snapshots forcing the project history to be captured and documented in an open forum.




Furthermore, with regards to the timeline of the project, the introduced checkpoints ensure a consistent synchronization of all complementary sub CRs. Subsequently, the multi-technology competence of the different teams is made available all along the timeline of the project. This is especially true for the integration phase.

## 6.5 Simplification of the process

As reported in section problems, the practitioners were complaining about the complexity of the process. In this section, we draw the shape of a solution susceptible of alleviating the increasing complexity of the development process at Ericsson.

We consider simplification rather than reengineering. The radical tailoring philosophy states that an improvement shall throw away all legacy practices and directly start with more efficient processes. However, this approach mostly fails because the organizations looses its key competencies which are substituted by a new model with another focus or because a model introduced from outside tries to radically change the work culture of the people.

Simplification is a more viable approach than reengineering for several reasons. In fact, simplification is easy to perform and impacts only the part of the process that is time consuming. Morover, simplification does not need specific training for engineers rendering it a cheaper solution compared to reengineering.

Evidences from literature studies (Dandekar, 1996; Dandekar, 1997) have shown that simplification of the process yields interesting results with regards to lead time. In (Dandekar, 1996) the authors present a case where simplification of the software process resulted in a saving of 20% in cost, 20% in human effort, 40% in elapse time and 30% reduction in the number of activities. These positive results motivate us to make use of this approach for the Ericsson Customer development organization.

To simplify the CR process, we identified prime candidates simplification:

* looping structures of iterative process steps; review of documents and code review with the aim of aligning the code to the Ericsson standard
* extensive documentation process:

### 6.5.1 Reducing review loops using real time review:

For reviewing a document, the modus operandi at EMP is as follows: after writing a document, the writer sends the document to the reviewer. The reviewer checks the quality of the document and issues remarks to the writer in order to upgrade and/or correct the document. The review is repeated a number of times until the document is correct.

To avoid this considerable waste of time spent in review loops we propose to introduce the notion of "real-time review". This notion is akin to XP pair programming. In the pair review practice, the writer of the document and the reviewer stand face to face with a first draft of the document to be reviewed.
In this form of review, the reviewer can raise questions about the document and the writer responds. If some correction is needed, it is performed instantaneously in a way that the reviewer can validate it. Such a review results in a continuous dialog between the pair leading to higher quality of the documents and avoids unnecessary loops.

The same idea can be applied for code review with the aim to align the code to Ericsson standard style. In fact, the reviewer and the code developer can held a meeting and the alignment of the code to Ericsson Standard style is done in a real time fashion. Therefore, loops and back and forth are avoided.

### 6.5.2 Simplifying the documentation process

The Agile literature uses the adjective "comprehensive" in its discounting of documentation. None of the Agile methods dispenses with documentation completely. Rather, they each seek to avoid wasting time and effort in producing documentation.

Documents are produced to communicate information. However, documents are not the only way of communicating information. The Agile methods advocates the use of face-to-face communication to the determinant of producing formal documents whenever possible, as it may be more straight to the point. The documents as a form of communication are limited by the amount of information they can convey, because they consist merely of words and have no other cues to help the reader understand them. Using documents as a primary communication mode on projects is problematic because of readers' penchant for misinterpreting the writer's intent. The value of a document lies in its persistence. On the opposite hand, the production of documentation can become a waste of time and effort if the documentation:

- Does not have a clear purpose; If there is not a clear purpose or use for the document, then producing it is likely to be wasteful.
- Does not have a clear audience; If the audience consists of multiple roles, then we should consider whether a single document can meet the needs of all of them. If there is not a clear audience, or if the audience is too diverse for the document to be practical, then producing it is likely to be wasteful.
- Is over engineered (or under engineered) for its purpose and audience; A minimum amount of effort should be used to ensure that the document can fulfil its purpose.
- Is maintained beyond its useful life; maintaining a document beyond the time when its audience can use it for its purpose is wasteful.
- Is recording the results of interpersonal communication (or prepare for it); Then it is likely to be wasteful, because words absent from the benefits for face-to-face communication are a poor communication mode.

At Ericsson, we reviewed the documents involved in a CR process. With the aid of some developers of the customer organization, we identified documents that can be simplified if instead of writing down information that does not require the persistence and therefore can be communicated face to face.

The module design specification is a document that contains a lot of text which can be communicated through meetings. This huge amount of text can be substituted with figures that can be explained vocally during sprint meetings.

Module test specification is also a candidate for simplification in the same manner.

## 6.6 Coordinated Scheduling of cross functional CRs

A delay in a delivery is the most unwanted situation at EMP. To meet customer satisfaction and fidelity, prompt delivery times is of paramount importance. Many CRs at EMP embrace different components of the EMP platform involving a broad technical expertise. Consequently, the CR is divided into complementary sub CR, each assigned to a specialized team responsible of an affected area. The main CR is only completed once all the related sub CR have been developed.

Despite the increasing importance of scheduling cross functional CR in the industrial field, only a small body of literature has explicitly addressed scheduling customer change orders. This problem of scheduling cross functional CR is akin to the assembly problem (Ahmadi et al., 1990).

This assembly problem is a novel scheduling problem that was first addressed by Ahmadi (Ahmadi et al., 1990) under the name Coordinated Scheduling of Customer Orders Problem. The core of the problem, is that the assembly can only be performed if all parts for assembly are available. A full solution of the problem was presented in (Ahmadi et al., 2005). As opposed to classical literature, Ahmadi supposes the machines are dedicated, meaning that one machine can only process one type of assignment. As EMP is specialized into teams each responsible of CRs within a technical domain (e.g. USB team, WLAN team etc.), the notion of machines is in perfect harmony with the organization of EMP Customer Development Team.

Further pursuing the principals of the assembly problem, customer orders should be processed in a specific order. This is where EMP Grimstad deviates from the best-practice findings of Ahmadi. There is no centralized dispatching of CRs to coordinate the handling order. The widespread practice at Ericsson is that the order of handling is left up to the concerned teams.

The shortcoming of this approach is that it looks at the complementary sub CR individually, and not from the customer perspective. In fact, the CR can not be completed and delivered until the entire corresponding sub CRs are completed as well. As a result, the completion time is mainly determined by the time at which the last cross functional sub CR is completed.

Therefore, to achieve better coordinated scheduling, the most beneficial approach is likely:

"...to shift our focus from performance measures inside a job shop (or company) to how it is perceived from the outside. Clearly, from the customer's perspective the arrival date of an order is highly relevant, whereas the speed at which the individual order components (jobs in classical jargon) pass through a job shop [or company] is only relevant to the extent that they serve this goal. For example, Shapiro, Rangan, and Sviokla (Shapiro et. al, 1992) describe a company which ships 99% of all order components on time, but only 50% of all customers receive their complete orders on time. Whereas the traditional inside performance measure of this company is quite respectable, its outside performance, as perceived by the customer, is marginal at best." (Ahmadi et al.; 2005)

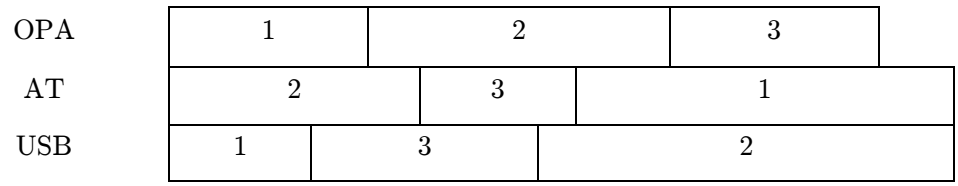
To map the problem as it is defined by Ahmadj (Ahmadi et al.; 2005) to Ericsson case we assimilate the dedicated machines as "specialized teams" (Like USB team, WLAN team, etc ..) teams, and the orders as cross functional CRs.

We consider an example of three cross/functional CRs: CR1 CR2 and CR3 Each CR is a composite of three sub-CRs that affects the Open Platform Architecture team (OPA), the AT-commands team (AT) and the USB team.

The estimation of the effort of each sub-CR is given in weeks in the following table.

| | OPA | AT | USB |
|---|---|---|---|
| CR1 | 3 | 7 | 2 |
| CR2 | 4 | 2 | 5 |
| CR3 | 2 | 4 | 6 |

We suppose that the OPA team handles the CRs in the following order: CR1, CR2, then CR3. Then, we suppose that the AT team handles their CRs in the following order: CR2, CR3, then CR1. Also, we suppose that the USB team handles their CRs in the following order: CR1, CR3, then CR2.



It is clear from the figure that the completion of every CR is delayed because one of its sub CR was last processed as last in the queue by one corresponding team. Therefore, the completion time of CR1 is after the AT finishes it after 7+2+4 = 13 weeks. Similarly, CR2 finishes it after the USB team finished it in 2+5+6 = 13 weeks. CR3 is finished when OPA team is finished after 3+4+2 = 9 weeks. Therefore, the average completion time is (13+13+9)/3 = 11,6666 weeks.

An application of the algorithm defined by Ahmadj (Ahmadi et al., 2005) will give the optimal strategy of scheduling.

Optimal completion time:
CR1: 9 weeks
CR2: 5 weeks
CR3: 13 weeks


In this example, the optimal scheduling results in an average completion time of 9 weeks, which illustrates the usefulness of the approach.

With respect to these findings, we have demonstrated that a customer perspective approach minimizes the average completion time of cross functional CRs. Moreover, the beginning of each Time Box offers a perfect opportunity for such an alignment of the teams' CR schedules. With the synergy of these approaches, a prominent part can be played in reducing lead time. We therefore strongly suggest that both solutions are adopted by the EMP customer organization and centralized management unit that can coordinate the handling order of each team's CRs

## 6.7 Software requirements prioritizing

Prioritizing of CRs at Ericsson is done by the CR owner when a CR is created. The different levels of priorities are ranked in the order “low”, “medium”, “high” and “critical”. Within these classifications there are no special order in which CRs are processed. It is usually by “first come - first served” principle. Ericsson needs to get a more efficient and accurate way of prioritizing software requirements. This has been highly acknowledged in the literature (Zave, 1995 ; Davis, 1993). In this perspective, we propose a using technique called the pair-wise comparison. This technique has been proved to outperform other prioritizing techniques (Karlsson, 1996). The pair-wise comparison technique is based on the analytic hierarchy process, AHP (Saaty, 1980). It requires the line-manager to consider the relative importance of all CRs and to what extent they differentiate, by comparing them pair-wise. This has several advantages over designating CRs with absolute priorities.

To make an assessment of the relative importance of two tasks, a scale is used as shown below. The scale is ranged from 1 to 9 according to how much *more* important one task is over another task. If a task is *less* important, the inverse value is used. E.g When task A has very strong importance over task B, then the relative priority is set to 7. Task B however, is very strongly less important than task A, hence the relative importance of task B over task A is 1/7.

The table below shows the grading of importance using pair wise-comparison. For instance, intensity of importance 5 means that a requirement is essentially strongly more important than another.

| Intensity of importance | Definition |
|---|---|
| 1 | Equal importance |
| 3 | Slight importance of one over another |
| 5 | Essential or strong importance |
| 7 | Very strong importance |
| 9 | Absolute importance |
| 2, 4, 6, 8 | Intermediate values between the two adjacent judgements |

When pair-wise comparing all n tasks against each other, their relative priorities are inserted into a comparison matrix of order n. For all the pairs,

the result is placed in the row and column representing the two tasks respectively. In the transposed location, the inverse value is inserted. The diagonal represents a task compared to itself. The result is naturally "Equal importance", hence the matrix main diagonal exclusively contains the value 1.

| | **A** | **B** | **C** |
|---|---|---|---|
| **A** | 1 | 7 | 4 |
| **B** | 1/7 | 1 | 1/3 |
| **C** | 1/4 | 3 | 1 |

Figure1 shows a comparison matrix with three tasks and their relative importance.

- Intensity of importance for task A over task B = 7 (very strong)
- Intensity of importance for task A over task C = 4 (medium strong)
- Intensity of importance for task C over task B = 3 (slightly more)

When the matrix is completed, the relative priority of each task is calculated.

## 6.8 Avoiding bottlenecks

In our in depth study of the problems at EMP, we have identified that the process lead time is dominated by the testing activity referred as the bottleneck activity. In this section, we propose two solutions to handle this problem. The first solution advocates a more balanced allocation of the testing resources. The second solution is related to the inherent nature of the agile methodology.

### 6.8.1 Pooling testing resources

When it comes to testing CRs, congestion is one of the main drivers of lead time at EMP. In fact, only two testers out of 35 are dedicated to test CRs. However this situation is controversial. In fact, during an interview with the manager of the customer development team EMP Grimstad he affirmed that half of the new functionalities introduced into the platform come from change requests.

This sharp division of testing resources between software release on one side and testing CRS on the other side has two reasons. The first reason is that at EMP there is an implicit trend to prioritize testing software release to the detriment of CR. The second reason is that the engineers may have to go through mental settings and become less productive if spread across different tasks, namely software release and testing CRs. The remaining testing resources are exhausted by software release tests.

As a solution we propose to use pooling as it is useful from a queuing perspective. The term pooling is used to describe the concept of flexible sharing of human resources. The resource to be pooled is the testing capacity which is measured by the number of engineers working as testers (for both software release and CR testing). The aim is to balance the utilization of the testing resources between software release and testing CRs proportionally to the magnitude of effort needed for each of the two activities. If this sense, if the CRs represent n % of the functionalities to be tested, then the testing team should consequently dedicate n % of it resources to test these functionalities.

The CR team and the software release can fairly share the testing resources. Instead of dedicating two persons to test CRs and to prioritize platform release tests, we render the utilization of the resource more flexible. Consequently, the testers will have a broader technical responsibility instead of "being specialized" in either testing software release or CRs.

To apply this in practice, the test department team should upgrade its allocation of human resources. Periodically, the test department should estimate the effort needed for testing CRs versus the effort needed for testing software release. Then resources should be allocated to each of them in a manner proportional to the magnitude of estimation.

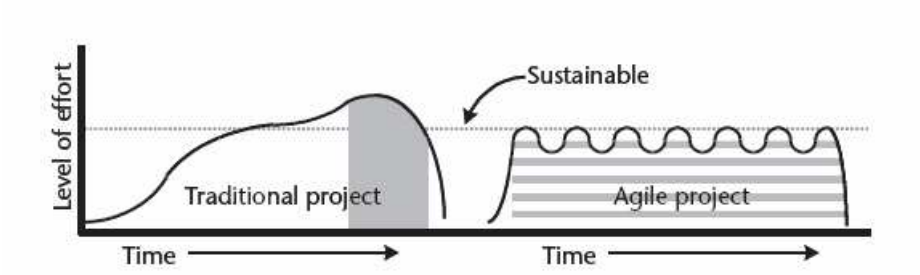
### 6.8.2 How agile can reduce bottlenecks

The Agile methods build a project environment that is likely to result in the project maintaining a sustainable pace (Larman; 2003). The Agile methods explicitly minimize the length of each increment. By keeping the time frame for each increment very short, the Agile methods help the team maintain a reasonable level of pressure almost continually. Since goals for each increment are relatively modest, the pressure is not excessive. Hence, the work that needs to be finished in that time frame is rarely overwhelming.

The short time frames also result in measurable achievements on a regular basis delivered in increments to testers. Thus, this sustainable pace of work for developers also results in a sustainable pace of work for testers.

Testing is generally the project phase that can stretch beyond expectations, resulting in forced overtime. The Agile method's small increments mean that only a limited amount of new functionality is being tested at any one time. With a limited scope, testing can be more easily managed, and fixing defects is less likely to become a bottleneck.

The figure below shows that in the Agile methods, testing is not a phase at the end of the project. Instead, it is an activity that the entire development team is engaged in throughout the life of the project.




## 6.9 Iterative defect analysis

Defect logging is a modus operandi at Ericsson. Data defects are stored in specialized data bases in a systematic fashion as soon as they are discovered. Nevertheless, defect logging is used at Ericsson only as a quality indicator. No analysis of defects is performed in order to prevent occurrence of the same defects in the future. Controversially, a mature software process is concerned with defects avoidance and reducing defects density.

In this perspective, Defect prevention is a viable technique that aims at reducing defects by dealing with their causes. Defect prevention starts from the premise that if an error has occurred, it will happen again unless something is done to stop it. A myriad of papers has been published in this context highlighting the merit of applying defect analysis. Empirical studies reported in an IBM Systems Journal article (May et al., 1990) concluded. "Reductions in

defects by more than 50% [were] achieved at a cost of about one-half per cent of the product area's resources". Pertinent evidence from case studies that defects prevention drives down error rate was presented by N. Card (Card, 1993).

Defect prevention has a twofold fruitful impact. First, it ameliorates the quality by decreasing the number of residual defects. Secondly, it fuels productivity by significantly saving additional effort spent on correction. This gives the engineers more time to spend on added value activities.

In practice, the deployment of a defect prevention strategy depends on the underlying development model (Jalote et al., 2005). For instance, the deployment varies from the case of a waterfall model to the case of agile development model. Jalote presents a study (Jalote et al., 2005) where he integrated Defect prevention into an iterative development model. The lessons drawn from the current iteration can be utilized in the next iterations. The aim is to create an adaptive scheme for defect prevention that responds quickly to defects as soon as they appear.

In our case study, since we advocate Scrum is iterative, we make use of this concept of defect prevention by moving the leveraging experience to the end of each Sprint loop. The strategy is to improve the quality and productivity in future Sprints by making use of the experience drawn from an earlier Sprint. As far as fine-tuning is concerned, in order to improve the effectiveness of defect prevention and its pragmatic feasibility, we intend to adapt it to the characteristics of EMP Customer organization. Instead of applying the technique at the overall customer organization level as done in the literature, we propose applying it at the team level. For example, the USB team and AT-team employ Defect prevention in separate meetings.

To enact this in practice, we propose to form two teams: causal analysis team and the action team.

- **Causal analysis team**

A Causal analysis team identifies defects and suggests actions that might prevent their occurrences. Members of the team should be the actual developers. The members meet periodically to analyze problems that arise and to discuss prevention of their recurrence. At the end of each iteration, the defect data is collected. The analysis of the defect data is performed to deduce the root causes of the most redundant defects that need special attention.. Root Cause Analysis (RCA) (Malinaric et. Al., 2000) is a technique appropriate for identifying the causes and inner mechanisms that lead to costly or risky problems related to the quality of the delivered products or the efficiency of the

development process. The principle of root cause analysis is to focus on causes for fault injection. If causes are eliminated, the defects will subsequently disappear. The goal of RCA is to formulate recommendations to eliminate or reduce the incidence of the most recurrent and costly errors in subsequent iterations. Then, design solutions to tackle the root causes to decrease the rate of defects. The team produces a list of prevention actions and suggest process improvement guidelines to avoid repeating same type of errors. These guidelines are forwarded to an action team.

One example of corrective actions is the following: The most frequent faults can be highlighted and workshops or training sessions are held to teach programmers how to avoid these faults. The premise of these workshops is that the programmer can learn faster from his repeated mistakes, and consequently gain a valuable time and effort by avoiding committing similar errors. In fact, if the most frequent errors create usually unnecessary rework that can be avoided if these errors are prevented.

Another example of corrective actions is the following scenario: A review of specification documents is held to include additional data aiming at covering the most frequently emerging issues.

- **Action team**

The action team prioritizes the suggested improvements and ensures their implementation. The action team needs authority and good communication skills, so management participation on this team is critical. Management support is especially helpful when suggested actions cross departmental or functional boundaries.

In a complex software system, such as EMP platform, developers spend an extensive amount of time correcting errors. By having a conscious strategy for preventing errors, the results of fewer errors injected and less necessary rework entails respectively higher quality and productivity. The main advantage of Defect Prevention is it low cost and straightforward simple deployment. However, its results can be spectacular.

## 6.10 Errors guessing

At Ericsson, determining the cause of an error is mostly a tedious and frustrating task due to the increasing complexity of the EMP platform. The EMP customer team consists of many young engineers that do not have much experience. Therefore, guessing the causes of reported errors can be a challenging task. This task is proven to be much easier for an experienced developer who is familiar with tracking down errors (Zapf, 1994a; Zapf, 1994b).

Supporting this, is a statement made during our interviews with one of the developers at EMP: "I spending one week struggling to find the cause of an assigned error report, which my experienced colleague helped me to pin-point in a matter of minutes". Hence, error guessing is a field that relies on experience and is best performed by senior developers of a company.

To reduce the lead time of handling error reports, we propose an innovative approach where Ericsson Grimstad forms an error guessing team consisting of a majority of senior members. The team's role is to identify possible causes for errors. All members of the error guessing team are experts trained on error diagnostics, able to identify the most common errors. The team meets regularly to provide first diagnostics of error reports to help developers identify causes and hereby help them correct it. The diagnostics of error reports contain a prioritized list of possible causes of the error. This diagnostics are communicated later to the developers to which the error report (ER) is assigned. Such a step can save valuable time by directing the developers to the exact point of interest.

To the best of our knowledge, this solution has not been presented before in the literature, so we would be pleased to see practitioners at EMP Grimstad put it into practice.

## 6.11 Modularization

Long build time is a pertinent problem at Ericsson. To reduce it a modularization is needed. However, this task can only be done by those responsible of the modules (Software platform release program). Due to limited resources, this task can not be deployed in a large scale. In an interview with a manager responsible of the software release he expressed that they are aware of this problem however due to limited resources, a corrective action can not be performed.

The problem mentioned here at Ericsson is akin to a problem reported by IBM. Under limited resources, IBM has prioritized the modules with high error rate as a prime candidate for modularization (Kaplan, 1994). This approach was known as high-risk module analysis. The approach has a twofold fruitful impact. First, it reduces the build time by breaking down modules into sub modules. Second, it reduces the error rate by dividing the most error prone modules. In this perspective, evidences from the literature have shown that

software modularization reduces the fault rate (Card et al., 1985; E.K. Emam et al. 2002; Kaplan, 1994).

We propose to use high-risk module analysis as a basis for modularization at EMP platform. According to Kaplan (Kaplan, 1994), the prime activity before modularization is to classify the modules into three categories; zero risk, low risk and high risk.

The Zero-risk category is allocated modules that had no errors during the previous iteration and that still do not have produced any errors.

The low-risk category is allocated modules that had less than average number of errors in either the previous iteration *or* after current iteration.

The high-risk category is allocated modules having more than average number of errors both after previous *and* current iteration.

High-risk modules are the prime candidates for modularization. High-risk modules are broken down into smaller modules as studies have shown that there is a correlation between the size of modules and frequency of errors. They are redesigned to reduce complexity.

## 6.12 Relocating testing

Changing the order of tasks in a software development process can have great benefits on quality and lead-time, and can be realized in most development processes.

One of the tasks that especially has proven to create benefit in such a manner is testing. Testing at Ericsson in Grimstad today conforms to the traditional model of having a team of testers, the System Verification team, separated from the developers. The developers develop the software modules, then optionally design and perform module tests before passing the result to System Verification for final testing. System Verification is responsible for running old scripts to verify that old requirements still pass the tests, as well as developing new test scripts to handle new requirements.

When the testers find bugs, they report the defects in Ericsson's error tracking system. Before a final product is accepted, it is common that the test-debug process is repeated several times.

There are at least three disadvantages to this approach; firstly, it makes quality mainly the responsibility of the testing team, not the development team that is making the product. Hence, the developers are not as concerned about quality as they should. Middleton claimed that "by moving responsibility

for measuring quality from the manger to the workers, a much quicker and more thorough response to defects was obtained“.

Another disadvantage is delay in communication between the teams. Communicating back and forth in order to resolve an error found by the testing team is a waste of time and effort.

Thirdly, having iteration loops of running tests and debugging at the end of the software development process, thereby passing defects back and forth, is inefficient especially when the location of the defect can be obscure.

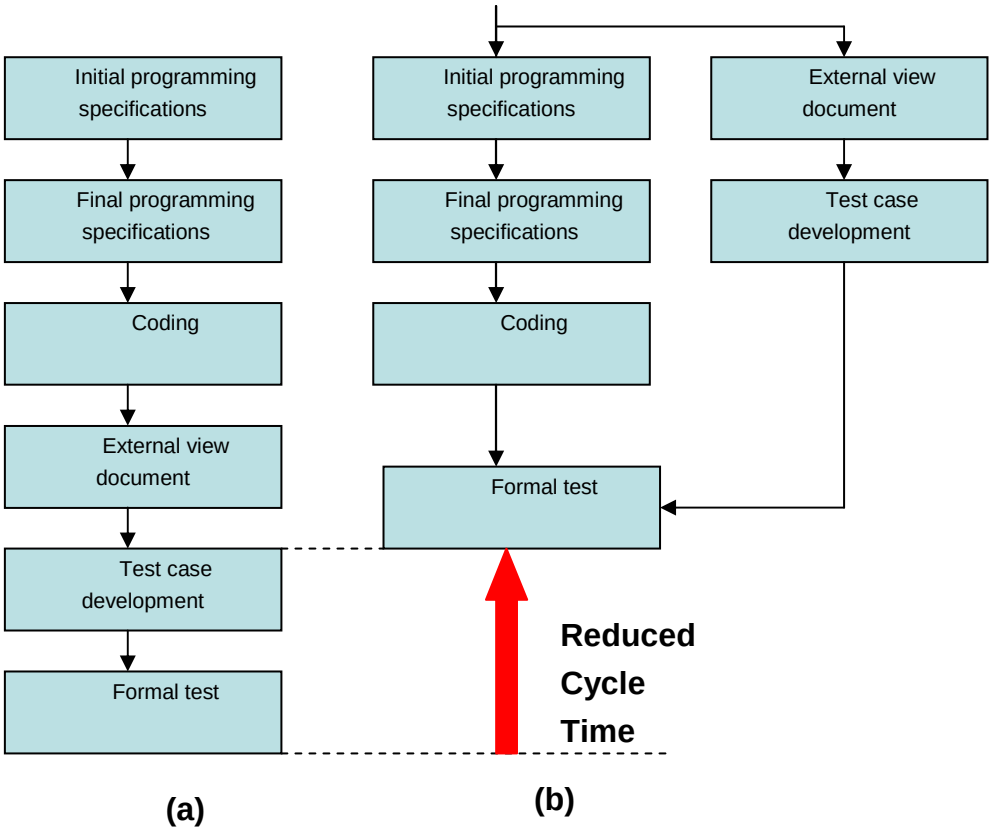
We propose a solution where we separate the roles of *test developers*, those who design and implement test scripts, and *test runners*. Because running the test scripts needs not necessarily be performed by the test script developers, it can in more extent be run by the software developers. This approach will decrease the workload for the test developers at Ericsson Grimstad who at the time being have limited resources, as they will get fewer errors to detect and report if many of them are already processed by other test runners like the developers.

Moreover, it is a well known fact within the field of software development that many defects are introduced by fixing othersTherefore it is of high importance that they are picked up at an early stage. This will give the software developers an instant feedback on if their changes have introduced any errors elsewhere in the system an it can save a lot of time compared to having to track down the error later.

A second improvement can be gained by simply shifting the implementation of module test *before* the implementation of the code. This facilitates an approach known as “test driven development”, and it can further alleviate recurrent testing and quality problems. As seen in the figure below, this approach reduces the lead time.

By starting development with writing test cases before new functionality is implemented, the developer gets focus on requirements and quality as well as module design as the developer initially must think about how the code is designed in order to be tested. In contrast, the opposite order usually makes the developer design the test in the same flow as he implemented the code, with the risk of repeating design / logical errors.

The process of Test Driven development is done in small increments where the developer makes a few test cases, implements the code until all tests run successfully before doing another increment with next set of functionality.




**The waterfall process. (a) before; (b) after**

## 6.16 Improving communication

Communication plays a decisive role in the efficiency and success of an organization. A variety of communication methods exists, each having different characteristics and richness. These attributes refer to the amount of information transmitted by a certain mode. The natural richness of speech is why Agile development, among others, advocates the use of "face-to-face" communication. The two-way communication of "face-to-face" opens for listeners to test their understanding. Where it is easy to misconstrue written words, voice adds a significant of clues and information. Additionally, two-way communication gives an opportunity of raising questions whenever there is need for clarification. These are the characteristics that make speech much more valuable than a document.

The use of tools can further enhance communication. In accordance with Agile practices, Ericsson Grimstad is frequently making use of whiteboards. Drawing a whiteboard illustration is a simple way of augmenting comprehension. In our

approach, we aim to take advantage of more of the before mentioned strengths and richness of different communication modes, by proposing tools that can improve communication at EMP:

### 6.16.1 Portal

As described in the problems section, one of the most recurring problems at Ericsson is finding information that at one point has already been passed among developers. Knowledge that is neither shared outside a small group nor documented can be lost. It usually requires substantial effort to find by those trying to acquire it.

As a tool to remedy this, we recommend a portal offering a discussion board to expand the area within knowledge is shared.

This idea is not new. It has been used since the advent of internet, as newsgroups and discussion forums. An identical approach on a local scale is perfect to gather the distributed knowledge and experience of it's users. The technical problem is raised on the forum and interested people try to propose solutions that assist or completely solve the question.

The threshold of making an entry in such a way is much lower as it is perceived as less informal. Just like e-mail is used today, but with the advantage that information is persistent and can be reviewed by new parties at a later stage.


Incidents reported in FIDO (the management system) should have their own threads so they can be discussed in the portal.

Whenever a developer faces a new problem, he can easily do a search in the database to see if the same problems have been solved before.

Many projects are of collaborate nature that require communication across departments. To facilitate getting in contact with the right persons, every employee should have a profile on the portal describing their experience, expertise, what project they are currently working on and scope of responsibility in addition to the usual personal data and photo.

### 6.16.2 Video conference

During our work at EMP we got to participate in a telephone conference meeting with a remote development site in Lund, Sweden. The two sites are both working on components used by the mobile platform, and therefore require a lot of coordination. Some of the meeting activities also require participants to travel to Lund as the limitations of telephone conference make it impractical as a real alternative.

The meeting was conducted by a group of employees from Grimstad who in advance had scheduled a voice conference with colleagues from the research and development departments in Lund. Prior to the meeting, microphones and loudspeakers are placed on the conference table.

We noticed that communication during telephone conference was noticeably inhibited compared to real-life meetings as it is missing the visual part that is so often used in non-virtual situations.

The setting may in some settings be perceived as uncomfortable as each side is unaware of all non-audible actions on the other location. In addition, the impression is enhanced by poor sound levels.

We recommend that EMP accommodates more use of virtual meetings as a mean of communication between sites. IBM has done a study (Kaplan, 1994) where they made use of video-conference at time when the technology was still immature and costly. The study found that just like in a real-life "face-to-face" conversation, video conferencing is more effective as it gives more information than a phone conference due to the added richness of body language.

Taking video conferencing one step further can be supplementing the video with a projection of a common computer screen on the wall. Illustrations can easily be added to augment communications and make comprehension rise considerably as well as act like thought-holders during the conversation. When key-persons attend such a meeting, real-time decisions and documentation can be made. Using a group computer with access to the common codebase and developer tools, even code changes can be done real-time. As the results are validated by all participants, quality is increased and the risk of failure to communicate is eliminated.

# 7 Experiment

The aim of the experiment is not to verify our ability to practice our technical skills in programming. Rather than this, the paramount objective is to assess the approaches we have proposed to verify their impact on the software development lead time. Consequently, we will not include all the low-level details of the technical solution, like the source code. This would by any matter not be disclosed due to our confidentiality agreement with Ericsson.

We will present the experiment in a qualitative- rather than quantitative approach.

## 7.1 Background

Mobile phone equipment is getting more and more complex, incorporating an increasingly number of functionalities. Mobile phones are becoming more and more sophisticated with advanced operating systems.

This development is posing key challenges for the developers. Users expect the mobile phone to have the processing power of a small computer, while still having the battery consumption of a calculator.

The technology of energy storage is not yet mature enough to support the increasing demand of processing power. Therefore, such a situation places an additional burden upon the software and hardware developers who need to optimize power consumption to the highest possible extent.

Consequently, Ericsson is dedicated to pursue every opportunity of reducing power consumption. A number of CRs filed in the incident management system are targeting power management.

## 7.2 Change Request Specification

The CR we have implemented originates from the need of further reducing power consumption when putting a mobile PC-card's USB-bus into suspend mode. Previous implementations of the platform have complied with the USB 2.0 specification (USB.org, 2002) which states that "all devices must support the Suspend state". However, in this state the standard only specifies how much suspend-current a device can draw from the bus itself. It is still up to the device to manage its internal power consumption. The consumption can be reduced by managing the power of the on-board hardware USB-block and transceiver. To facilitate this, the EMP hardware developers have made an

upgrade to the Application Specific Integrated Circuit (ASIC). The upgrade makes it possible to control the USB transceiver power modes through software. The purpose of the CR is to implement the software part that interfaces the power management hardware.

## 7.3 Technical environment

Ericsson Customer Development Team are the developers of key modules for the mobile platform software. The technical environment created to perform this task, consists of the SDK tools (compiler, linker, builder (SDE) etc), a code management system (CME) and the platform assistant, which is the software to sign and flash the mobile device with the software product. To support the process, there are additional tools like software debug simulator and hardware debugging, an incident manager (FIDO) where all change requests and error request are filed, and a document repository system.

### 7.3.1 Incident report and status tracking tool

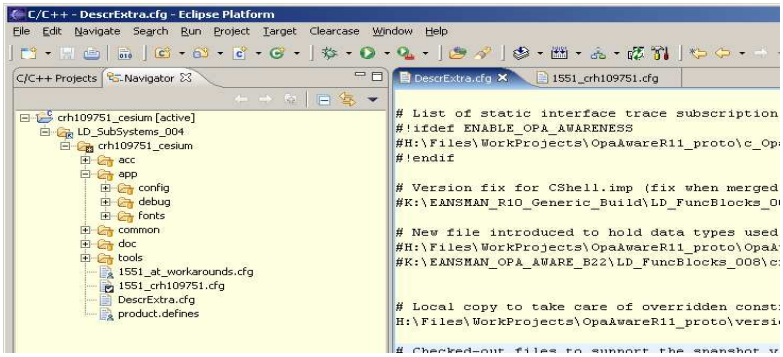
The tool is used to track reports originating both from customers and EMP internally. Incident reports (IR) are kept in the system together with the people or groups that are involved in their investigation. The tool is used by a variety of people within EMP (and also external customers), e.g. developers and managers to report and check the status of tasks.

### 7.3.2 Source code management system

The source code management system at Ericsson is based on IBM's Clear Case, a software tool for revision control. It utilizes its own interface adapted to the development processes at Ericsson. The source code management system is the entry point when starting on a task. It is used to create a configuration with the targeted module, and its belonging source files. From here the developer can start the compiler.

### 7.3.3 Source code editor

Eclipse is an Integrated Development Environment (IDE). It includes a source code editor and a number of useful tools to edit source code. EMP's build environment SDE is included through plug-ins.




### 7.3.4 Platform Assistant

Platform Assistant is tool used to access mobile phone hardware. The tool enables the developer to connect to prototype phones through USB or RS232. Once the platform software is built from Eclipse, Platform assistant is used to sign and transfer the binary files into the phone Flash memory.

## 7.4 Applied solutions

We intend to include a selection of our recommended approaches when implementing this CR to validate their efficiency. The choice of solutions is

affected by the nature of the CR. As the CR is a challenge technically, but does not cover a large scope with regards to management, this singles out the following approaches that we will include:

- Sprint – an agile approach
- Communication
- Simplification
- Error guessing
- Relocating testing

### 7.4.1 Sprint

The Scrum Agile development process is a software development methodology that aims at achieving short lead times. Scrum, is an example of tight process which maintains a high pace of productivity through successive delivery in short time boxes. The beauty of Sprint is that it makes use of Time boxing to increase the productivity of the team by imposing fine short time deadlines. In fact, we applied this principle dividing the CR into three main increments. We fixed a time box of one week to finish every increment as in the following plan.

1st week: Implement design and source code
2nd week: Adapt platform framework to accept parameters
3rd week: Modify build files, integrate in product.

Defining fine deadlines for each increment was shown to be useful. In fact. In the end of the second week we were late. So, we decided to work overtime and get the functionality achieved as close as possible to the schedule. By keeping the time frame for each increment very short, we maintained a reasonable level of pressure almost continually. This ended up with the completion of the development of the CR in the end of the third week.

### 7.4.2 Communication

Agile development in general emphasises the use of face to face communication. To gain insights into the development activities, we held daily sprint-interactions with the technical staff involved in USB and power consumption issues. This interaction played the same role in the process as sprint daily meeting. Obviously, communication is directly measurable to lead time. However, we felt that this was very time saving by getting core

information and avoiding wasting much time on irrelevant documents entailing extensive details.

In accordance fulfil the premises if our solution, we gathered a list of key persons in advance. During implementation we did not need to waste time to get to the correct person, thus saving a lot of time.

### 7.4.3 Error guessing

We made an error by calling the suspend function using wrong parameters. We spent two days looking for the error because the test failed. Then we decide to consult an experienced developer. He pointed out possible cause of error by saying that you might have called wrongly the suspend function or you modified wrongly the file containing the configuration of the USB module. Based on these two recommendations, we identified the source of error in the next minutes after we talked to him. In fact, we made a wrong parameter call of the function suspend.

Here we underline that the solution of introducing a team for guessing errors can be very useful to reduce the time spent in tracking errors.

### 7.4.4 Simplification

Code review, with the aim of aligning the code to the Ericsson standard style is a usually time consuming activity that is done in loops.

In the section solutions, we proposed real time reviewing of code review as a viable solution to avoid loops and back and forth between the reviewer and the developer. We implemented the strategy as defined in the solution. In fact, while one of us was implementing the code, the other spent some time to learn the Ericsson code style in details.

The code review was held in the end of the development as real time review. This significantly reduces the overhead of multiple iterations.

### 7.4.5 Relocating testing

The use of relocated testing as described in our solutions, gave us a goal oriented focus. As soon as the implementation design was finished, one of us wrote a test while the other implemented the code.

The test checked if the power-saving bit in the register was correct at all phases of suspend-resume operations. The goal of the implementation was to

pass this test. This way, we saved the time of having to design and perform the test in retrospect of implementing code.

## 7.5 Conclusion

The results are not conclusive as comparisons are impossible with just one sample. In fact, to verify our findings we should enact our proposed approach at a higher level that encompasses management processes, sampling change requests in a larger scale. This is why we could not apply more of the solutions within the limited scope of the current setting.

Not enough resources to perform a larger CR as it demands the involvement of an entire software development team. However, the results seem to be promising. Despite we are students and thereby novices to the software architecture used at Ericsson Mobile Platforms, we have achieved a notable gain in performance with regards to development lead time.

By applying more of the proposed solutions on the management level we are confident that we would be able to utilize more of the potential benefits they offer. This experiment by itself is not enough to conclude statistically significance, but implies that our approach should be run in a larger scale pilot project to be confirmed.

# 8 Discussion

In the current study we have addressed the problem of improving the software process at EMP customer organization in Grimstad. More specifically, we have investigated the question of how to reduce the lead time without jeopardizing the quality.

To proceed, we adopted a bottom-up approach. The premise of such approach, when applied to Software Process Improvement, is that we first should understand the existing problems before improving the process. Specifically, when mapped to our research, this meant first identifying the main drivers of lead time, before creating solutions to alleviate them. This approach is known as the problem driven approach within the area of Software Process Improvement. It allowed us to target specific problems within the process. An alternative solution was the norm driven approach, which is a strategy to adopt best-practice approaches presuming this will alleviate any problems. We did not use this as it implies that specific problems are never pin-pointed.

Being aware that the technical staff knows best the characteristics of the company, our research relied on interviews to support the literature study. In addition, we considered quantitative data collected from errors reports, change requests and similar statistics.

Our work to reduce the lead time initially resulted in the identification of 8 specific problems. We classified them as the main drivers of lead time when handling Change Requests at EMP. Our further work resulted in an extensive list of solutions that together address the problem in all aspects. It is a study of the relationship between Change Requests and lead time which attacks the problem in a way that has not been previously presented in the literature.

In the beginning of the project, we investigated the possibility of reducing lead time on the lowest level by removing overhead within the specific development tasks. Our findings showed that the EMP already had put a lot of effort into removing overhead. Nevertheless, as described in the solutions section, we have successfully identified some areas where the work tasks of the developers can be done more efficiently. We could have pursued this path even further. However when moving on with the project, literature study showed that the significant improvements within software development reside in the utilization of development methodologies. We closely examined these to support our approaches on reducing lead time. Subsequently, many of our recommended solutions are based on actions taken on a higher management level.

The experiment could have had a larger scope to validate the evidences we have presented from literature. However, putting the higher management level solutions into action could not be done within the time span of our project. They require the involvement of resources from management, multiple developers and other development departments. Resources that are already scarce since Ericsson have been facing an upcoming platform release. In addition, a simple Change Request can take months to complete.

Also, we have an extensive list of solutions. The solutions can not be adopted all at once as it would be impossible to differentiate which solution has the most effect.

As a future work, we propose a migration phase that would involve two phases:

-Introduction within one selected project: The best migration approach for EMP would be to initiate pilot projects utilizing a limited number of solutions at the time and collecting metrics during progress.

-Deployment and widespread adoption: Successful completion of the pilot test phase means the organization is ready to use the new method.

The success of these phases involves the participation of an expert group in Software Process Improvement deployment, tracking and planning. This group will be the driving force to direct the deployment of the migration activities and guide them with the cooperation of the Ericsson management. Depending on the results of the pilot project, the expert group has to define and agree, in cooperation with the affected middle management, a detailed deployment plan for the improvement actions deciding which improvements will be applied thoroughly, which improvements will be applied partially in specific software areas or in specific software teams and which improvements are not ready to be applied and have to be studied more.

In parallel to the definition of the deployment plan, a training plan for the developers impacted by the new methodologies has to be defined.

# 9 Conclusion

In this study, we investigated the problem of shortening the lead time of handling customer requests at EMP Grimstad. Deficiencies were first identified and an extensive set of solutions was proposed. A comprehensive experiment was conducted, and the results are promising. As future work, we propose to derive a general model of handling customer requests based on our results. Such a model can fill the void in the domain of improving the software process of handling customer change requests.

# 10 References


**A**

Aaen, I., P. Bøttcher, et al. (1998). The Software Factory: Contributions and Illusions. 6th European Conference on Information Systems (ECIS), Aix en Provence, France

Aaen, I., Arendt, J., Mathiassen, L., & Ngwenyama, O., 2001. A conceptual MAP of software process improvement. Scandinavian Journal of Information Systems, 13.

Ahern, D. M., A. Clouse, et al. (2001). CMMI Distilled: An Introduction to Multi-discipline Process Improvement. Addison Wesley.

Ahuja, M.J., Carley, K.M. 1998. "Network Structure in Virtual Organizations", Journal of Computer Mediated Communication, 3, 4. Available at: http://www.ascusc.org/)

Alec Dorling, "Software Process Improvement and Capability determination," Software Quality Journal, Vol. 2, No. 4, December 1993, pp. 209-224.

Ashok Dandekar and Dewayne E. Perry. "Barriers to Effective Process Architecture", Software Process: Improvement & Practice 2:1 (March 1996). pp 13-20.

Ashok Dandekar, Dewayne E. Perry, Lawrence G. Votta: Studies in process simplification. Software Process: Improvement and Practice 3(2): 87-104 (1997)

Arent, J., & Nørbjerg, J. (2000). Software process improvement as organisational knowledge creation—a multiple case analysis. Proceedings of the 33rd Hawaii international conference on system sciences. Hawaii: Wailea.

**B**

Bach, J. (1994). "The Immaturity of the CMM." American Programmer 7(9): 13-18.

Balcerak K.J. , B.G. Dale, Engineering change administration: the key issues, Computer-Integrated Manufacturing Systems 5 (2) (1992) 125-132.

Basili, V., Caldiera, G., & Rombach, H. D. (1994a). The experience factory. Encyclopedia of software engineering—2 Volume Set. New York: Wiley.

Basili, V., Caldiera, G., & Rombach, H. D. (1994b). The goal question metric approach. Encyclopedia of software engineering—2 volume set. New York: Wiley p. 528–532.

Basili, V., Caldiera, G. (1995). Technical Report: CS-TR-3483, UMIACS-TR-95-67, Univeristy of Maryland, College Park, MD 20742, USA.

Bhuiyan Nadia; Gregory Gatard; Vince Thomson Engineering change request management in a new product development process, European Journal of Innovation Management, Volume 9, Number 1, 2006 , pp. 5-19(15)

Boehm, B.W. (1984), Software Engineering Economics, IEEE Trans. On Software Engineering, 10(1), pp. 4-12.

Boehm, B.W. (1988), A Spiral Model for Software Development and Enhancement, IEEE Computer, 21(5), May, pp.61-72.

Boehm, B. and P. Bose (1994), A Collaborative Spiral Software process Model based on Theory W, Proc. 3rd International Conference on the Software Process, IEEE Computer Society Press, Reston, VA, October, pp.59-68.

Bollinger, T. B. and C. McGowan (1991). "A critical look at software capability evaluations." IEEE Software 8 (4): 25-41.

Byrnes P, Philips M. 1996. Software Capability Evaluation Version 3: Method Description. SEI Technical Report CMU/SEI-96-TR-002.

**C**

Card, D.N.; Page G.T.; McGarry, F.E.: Criteria for Software Modularization. Proceedings of the 8th International Confenrence on Software Engineering, August 28-30, London, 1985, pp. 372-377

Card D.N., Defect-causal analysis drives down error rates, IEEE Software 15 (1993) (1), pp. 88–89.

Cattaneo F., A. Fuggetta, and D. Sciuto, "Pursuing coherence in software process assessment and improvement", CEFRIEL, Milano, Technical Report September 1998.

Curtis, B., Krasner, H., Iscoe, N. 1988. "A field study of the software design process for large systems", Communications of the ACM, 31, 11, pp. 1268-1287.

Cusumano, M. A. (1989). "The Software Factory: A Historical Interpretation." IEEE Software.

Curtis, B., H. Krasner, V.Y. Shen, and N. Iscoe (1987), On Building Software Process Models under the Lamppost, Proc. 9th International Conference on Software Engineering, IEEE Computer Society Press, Monterey, CA., pp. 96-103.

Curtis, B., Krasner, H., Iscoe, N. 1988. "A field study of the software design process for large systems",
Communications of the ACM, 31, 11, pp. 1268-1287.

**D**

Dalcher D., Benediktsson O., and Thorbergsson H., "Development Life Cycle Management: A Multiproject Experiment", Proceedings of the 12th International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), 2005.

Davis, A. M. (1993). "Software Requirements: Objects, Functions and States". Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.

David H. Kitson, “An Emerging International Standard for Software Process Assessment,” Proceedings of the Third IEEE International Software Engineering Standards Symposium and Forum, Walnut Creek, CA, 1-6 June 1997, pp. 83-90.

David C. Carr, Ashok Dandekar, and Dewayne E. Perry. “Experiments in Process Interface Descriptions, Visualizations and Analysis”, Software Process Technology — 4th European Workshop, EWSPT’95 Wilhelm Schaefer, ed. Lecture Notes in Computer Science, 913, Springer-Verlag, 1995. pp 119 - 137.

Dewayne E. Perry. “Issues in Process Architecture”, Proceedings of the 9th International Software Process Workshop — The Role of Humans in the Process, October 1994, Airlie VA. IEEE Computer Society Press. pp 138 - 140.

Dean Leffingwell, Scaling Software Agility: Best Practices for Large Enterprises, Addison Wesley

Dunaway DK, Masters S. 1996. CMMSM-based appraisal for internal process improvement (CBA IPI): Method description. SEI Technical Report CMU/SEI-96-TR-007.

**E**

Emam E.K., S. Benlarbi, N. Goel, W. Melo, H. Lounis and S.N. Rai, The optimal class size for object-oriented software. IEEE Transactions on Software Engineering (2002), pp. 494–509.

**F**

Fenton N., Software metrics: A rigorous approach, Chapman and Hall. 1991.

**G**

Gilb, T. (1988), Principles of Software Engineering Management, Addison-Wesley, Reading, MA.

GMOD (1992), V-Model: Software Lifecycle Process Model, General Report No. 250, German Ministry of Defense.

Grady, Practical Software Metrics for Project Management and Process Improvement, Prentice-Hall, Englewood Cliffs, N.J., 1992.

Gustavsson, A. (1989), Maintaining the Evaluation of Software Objects in an Integrated Environment, Proc. 2nd International Workshop on Software Configuration Management, ACM, Princeton, NJ, October, pp.114-117.

**H**

Hansen B., J. Rose, et al. . Prescription, Description, Reflection: The Shape of the Software Process Improvement Field. International Journal of Information Management

Harrington, H.J. 1991. Business Process Improvement.; McGraw Hill, New York.

Hersleb J.D., Grinter, R.E. 1999. "Architectures, Coordination, and Distance: Conway's Law and Beyond", IEEE Software, September/October, pp. 63-70.

Hersleb J.D., Grinter, R.E. 1999. "Architectures, Coordination, and Distance: Conway's Law and Beyond", IEEE Software, September/October, pp. 63-70.

**I**

Ilieva S., Ivanov P., and Stefanova E., "Analyses of an agile methodology implementation," Proceedings 30th Euromicro Conference, 2004, IEEE Computer Society Press, pp. 326-333.

Imai, M., Kaizen, The key to Japan's Competitive success, Random house, New York, 1986.

Iversen, J., Nielsen, P. A., & Nørbjerg, J. (1999). Situated assessment of problems in software development. The DATABASE for advances in information systems, 30(2).

**J**

Jalote, P., Agrawal, N. 2005. Using defect analysis feedback for improving quality and productivity in iterative software development, in: Proceedings of the Information Science and Communications Technology (ICICT 2005), pp. 701-713.

Jarzombek S., Proc. Joint Aerospace Weapons Systems Support, Sensors and Simulation Symp., Gov't Printing Office Press, 1999.

Jones, T.C., 1994. Assessment and Control of Software Risks. Prentice Hall, NY.

Jorma Taramaa, Munish Khurana, Pasi Kuvaja, Jari Lehtonen, Markku Oivo, Veikko Seppänen: Product-Based Software Process Improvement for Embedded Systems. EUROMICRO 1998: 20905-20912

Jussi Ronkainen, Jorma Taramaa, Arto Savuoja: Characteristics of Process Improvement of Hardware-Related SW. PROFES 2002: 247-257

**K**

Kaplan, Craig, Ralph Clark, and Victor Tang. Secrets of Software Quality, 40 Innovations From IBM. McGraw Hill, 1994.

Konrad, M., M. B. Chrissis, et al. (1996). "Capability Maturity Modeling at the SEI." Software Process - Improvement and Practice 2 (1): 21-34.

Karlsson, Joachim, "Software Requirements Prioritizing" - SoftLab ab 1996

Kraut, R.E., Streeter, L.A. 1995. "Coordination in Software Development", Communications of the ACM, 38, 3, pp. 69-81.

Kraut, R.E., Streeter, L.A. 1995. "Coordination in Software Development", Communications of the ACM, 38, 3, pp. 69-81.

L

Larman C., Agile and Iterative Development: A Manager's Guide , Addison-Wesley, 2003

Layman L., Williams L., and Cunningham L., "Exploring extreme programming in context: an industrial case study", Agile Development Conference, 2004.

Lehman, M.M. (1985), Program Evolution: Processes of Software Change, Academic Press, London.

M

Malinaric D., R. Hoffmeistern, and C. Sun, "Case Study for Root Cause Analysis of Yield Problems," Proc. IEEE/SEMI Advanced Semiconductor Manufacturing Conf. and Workshop (ASMC 00), IEEE Press, 2000, pp. 8-13.

Mandal J.K.; Sinha A.K.; Wright I.C.1 A review of research into engineering change management: implications for product design, Design Studies, Volume 18, Number 1, January 1997 , pp. 33-42(10)

Mann C. and Maurer F., "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction", Agile Development Conference, 2005.

Matsumoto, Y. (1981). SWB System: A Software Factory. North-Holland,Amsterdam.

Matsumoto, Y. (1987). A Software Factory: An Overall Approach to Software Production. IEEE.

Mays R.G., C.L. Jones, G.J. Holloway, D.P. Studinski, “Experiences with Defect Prevention”, IBM Systems Journal, Vol 29, No. 1, 1990

McDermid, J.A. ed. (1991), Software Engineer’s Reference Book, Butterworth-Heinemann Ltd., Oxford, UK. Meek, B.L. (1991), Early High-Level Languages

McFeely R. 1996. IDEALSM: A User’s Guide for Software Process Improvement. SEI Handbook CMU/SEI-96-HB-001, February.

Mills, H.D., M. Dyer, and R.C. Linger (1987), Clean room Software Engineering, IEEE Software, 4(5), Sept., pp.19-25.

O

Ohlsson, N. and Alberg, H. 1996. Predicting fault-prone software modules in telephone switches, IEEE Transactions on Software Engineering 22(12): 886–894.

P

Parnas, D.L. (1979), Design Software for Ease of Extension and Contraction, IEEE Trans. Software Engineering, 5(3), March, pp. 128-138.

Palmer, J.W., Speier, C., 1998. “Teams: Virtualness and Media Choice”, Procs. of the 31rd Hawaii Int.

Palmer, J.W., Speier, C., 1998. “Teams: Virtualness and Media Choice”, Procs. of the 31rd Hawaii Int. Conference on System Sciences.

Pentti Marttiin, Jari A. Lehto, Göte Nyman: Understanding and Evaluating Collaborative Work in Multi-site Software Projects - A Framework and Preliminary Results. HICSS 2002: 23

Paulk MC, Curtis B, Chrissis MB, Weber CV. 1993. Capability Maturity Model for Software, Version 1.1. Pressman, R.S. (1992), Software

Engineering: A Practitioner's Approach, 3rd ed., McGraw-Hill International Editions, New York.

Peter Schuh, Integrating Agile Development in the Real World, Publisher: Charles River Media

Poppendieck M. and Poppendieck T., Lean Software Development - An Agile Toolkit for Software Development Managers. Boston: Addison-Wesley, 2003, ISBN 0-321-15078-3

Pressman, R.S., 2001. Software Engineering: A Practitioner's Approach. McGraw-Hill, NY

Pulford, K., A. Kuntzmann-Combelles, et al. (1992). A Quantitative Approach to Software Management: The ami Handbook. CSSE South Bank University, London.

R

Reifer, D. J. (2002). "The CMMI: it's formidable." The journal of Systems and Software 50.: 97-98. Guest editor's corner.

Royce, W.W. (1970), Managing the Development of Large Software Systems: Concepts and Techniques, Proc. WESCON, August, USA.

S

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

Shapiro B.P., V.K. Rangan, and J.J. Sviokla, Staple yourself to an order, Harvard Business Rev 70(4) (1992), 113–121.

Stelzer D. , W. Mellis, and G. Herzwurm, "Software Process Improvement via ISO 9000? Results of Two Surveys Among European Software Houses," Software Process: Improvement and Practice, Vol. 2, Issue 3, September 1996, pp. 197-210

T

Takeuchi, H., and I. Nonaka, “The New Product Development Game,“ Harvard Business Review, January 1986

Tanaka T., Sakamoto K., Kusumoto S., Matsumoto K., and Kikuno T., Improvement of Software Process by Process Description and Benefit Estimation, Proceedings of the 17th International Conference on Software Engineering, ACM, 1995, pp. 123-132

Terwiesch, C., Loch, C.H. (1999), “Managing the process of engineering change orders: the case of the climate control system in automobile development”, Journal of Product Innovation Management, Vol. 16 No.2, pp.160-72.

Tore Dybå, Torgeir Dingsøyr, “Empirical Studies of Agile Software Development: A Systematic Review”, Information and Software Technology (2008)

U

(USB.org, 2002) Universal Serial Bus Specification Revision 2.0 - Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V.

V

Vanhanen, J, Kähkönen, T. & Jartti, J. Practical Experiences of Agility in the Telecom Industry. XP 2003, May 25-29, 2003, Genova, Italy.

Virpi Taipale, Jorma Taramaa: Process Improvement Solution for Co-design in Radio Base Station DSP SW. PROFES 2005: 16-28

W

Watts F., Engineering changes: a case study, Production and Inventory Management Journal 25 (Part 4) (1984) 55-62.

Z

Zapf, D., & Reason, J.T. (1994a). Human Errors and Error Handling. Applied Psychology: An International Review, 43, 427-432.

Zapf, D., Maier, G.W., Rappensperger, G., & Irmer, C. (1994b). Error detection, task characteristics, and some consequences for software design. Applied Psychology: An International Review, 43, 499-520.

W

Westland, J. Christopher - The cost of errors in software development: evidence from industry, Journal of Systems and Software, Volume 62, Issue 1 (May 2002)

Wong C., "A Successful Software Development," IEEE Trans. Software Eng., no. 3, 1984, pp. 714-727.