UNIVERSITY OF GOTHENBURG

Lean Software Development

Theory validation in terms of cost-reduction and quality-improvement

Bachelor of Science Thesis in Software Engineering and Management

Christos Svitis

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, July 2013

Lean Software Development
Theory validation in terms of cost-reduction and quality-improvement

Christos Svitis

Examiner: Lars Pareto

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, July 2013

Abstract

The pace of change in software development industry remains at high. People continue to push the boundaries of known techniques and practices in an effort to develop software as efficiently and effectively as possible. Lean software development has emerged as an alternative to comprehensive methods designed primarily for very large projects. Key objective of Lean is fast development and delivery of a high quality system at a relatively low investment cost. In this study, an investigation on Lean's validity was conducted, by examining several organizations which have been developing software according to Lean's thinking. The outcome of this research was the validation of the cost-effectiveness of Lean and of its impact on the quality of a software system.

Keywords: Lean software development, cost-reduction, quality-improvement, validation

1. Introduction

This section gives a brief overview of the subject and theme of this study. It presents the background and the problem domain of the topic under study, and describes the purpose of this thesis together with a definition of the research questions.

1.1. Background

Every software development organization is using a process for the development of its products. In most cases, the process is adapted to each organization's size, resources, and needs, but the core characteristics are based on one (or more than one) of the "standard" development methodologies. "A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system." [1] However, statistics have shown that large number of software projects does not meet their expectations in terms of functionality, cost, or delivery schedule [29]. This is mainly owed to the "rigidity" of the traditional development processes, and their inability to effectively deal with the various challenges of today's software industry. Therefore, more and more companies are looking for better alternatives to improve the software quality, reduce the development cost, and meet the market demands and customer satisfaction. Lean software development emerged as an alternative to document-driven, rigorous traditional development approaches. Lean's philosophy is reducing the development time by removing all nonvalue-adding wastes. Lean thinking principles are based on the Toyota Production System [2], and have been successfully applied in many manufacturing and product development organizations. During the last few years, Lean develop-ment has also become popular within the software industry. Its popularity is due to its effectiveness in identifying and eliminating waste, and quickly responding to changing customer and market demands.

1.2. Purpose

There is a growing body of literature on Lean software development, with Poppendiecks' book [3] being the "cornerstone" of Lean's theory. Mary and Tom Poppendieck tailored the principles of Toyota's production process [1] to fit in the software engineering domain, and introduced Lean as a software development process. Thereafter, influenced by their work [3] [4], a large body of knowledge on Lean has become available. Several researchers have discussed advantages and disadvantages of Lean in relation to more traditional software development processes [5] [6], while others have focused on identifying limitations and problems associated with Lean implementations [7] [8]. However, despite the variety of literature about Lean software development, one that confirms its functionality is hard to be found.

The purpose of this study is to assess the applicability of Lean's theory in real situations. In other words, to investigate if Lean in practice has the results that the theory promises (similarities and differences between Lean 'in theory' and 'in practice'). The research was divided in two phases.

The first phase was an exploration of Lean's theory and its thinking principles. This was done by reading existing literature – especially the ones by Poppendieck [3] [4] – on Lean's theory. That helped in obtaining a deeper understanding of the theoretical assumptions of Lean and of its seven principles. In this, the cost- and quality-related principles/aspects of Lean were of primary interest, and hence focus was put on these. The knowledge gained from this phase, formed the theoretical "ground" for the qualitative analysis of the second phase.

The second – and most crucial – phase was to validate the theoretical findings by comparing them with results from Lean implementations. Data regarding results of Lean's implementations was collected in a series of interviews and a case study review from various organizations that have implemented Lean as their development process. The results from the case study review and the interviews were used to validate if Lean's theory is successful in real world applications, and the success rate thereof.

1.3. Research Questions

In order to address the above objectives, two sets of research questions were formulated to drive through the research process. As Creswell [9] describes, the research questions serve to narrow and focus the purpose of the study.

Research questions of phase 1 (literature review):

- How can Lean software development (theoretically) reduce the cost of a software product?
- How can Lean software development (theoretically) improve the quality of a software product?

Having these questions in mind during the review of related books/articles, helped in identifying the theoretical aspects of Lean related to cost-reduction and quality-improvement. This gave a deeper knowledge and understanding of Lean's theory and of its thinking principles, and set the theoretical basis of the second phase.

Research questions of phase 2 (interviews & case study review):

- Can Lean software development (in practice) reduce the cost of a software product, and how?
- Can Lean software development (in practice) improve the quality of a software product, and how?

The answer to these questions was the outcome of the comparison between the theoretical assumptions of Lean and the results of the data analysis, and is being presented in Sections 5 & 6.

1.4. Document Outline

The paper is structured as follows: Section 2 gives an overview of the theoretical frameworks used in this study (Lean software development theory). Section 3 describes the research design and methodology used in this study, while in Section 4 the results of the study are presented. Section 5 analyzes and discusses the findings and Section 6 concludes the work.

2. Theoretical Framework

This section gives an overview of Lean software development. It presents the history, the theory, and the thinking principles of Lean.

2.1. Lean's History

Lean's history starts with Lean production. In order to be able to know and comprehend the nature or meaning of Lean production, one first has to understand the concept of Lean.

It is accepted as true that Lean was first introduced in Japan - mainly in Toyota Production System - but history also shows that Henry Ford had been applying parts of Lean in the year 1920. "One of the most noteworthy accomplishments in keeping the price of Ford products low is the gradual shortening of the production cycle. The longer an article is in the process of manufacture and the more it is moved about, the greater is its ultimate cost." [10]

In the end of 19th century, the concept of Lean production started with a need of mass production system. Frederick Winslow Taylor on his time first thought about the high production system known as mass production, and it was presented in his book "The principles of Scientific Management" [11].

After the Second World War, an engineer from Toyota named Eiji Toyoda visited the Ford Plant. He noticed that everything was functioning smoothly, but there were wastes in essence but not in concept [12]. Eiji Toyoda observed waste everywhere in Ford's process. Afterwards, Eiji Toyoda with Taiichi Ohno (a production manager from Toyota) recognized that Ford production system was not going to work for Japan, due to, but not limited to, Japan's infrastructure. They started to improve Ford's production system, in order to make it applicable for them [13]. Afterwards, Toyota developed the process, which is now known as Toyota Production System. Taiichi Ohno explained the concept as "All we are doing is looking at the timeline from the moment the customer gives us an order to the point when we collect the cash. And we are reducing the timeline by removing the non-value added wastes." [14] Toyota changed the system and did continued improvements; however, it took almost 30 years for Taiichi Ohno to make it perfect for Toyota and make it as it is today.

Lean Production System is mainly described in the book "The Machine that Changed the World" by James Womack, Daniel Jones, and Daniel Roos [12]. In that book, the authors mention that Lean Production System by contrast combines the advantages of craft and mass production. In the time being, it also avoids high cost and rigidity of production by planning, therefore, it is known as 'Lean' (because it uses everything less). If it gets compared with massive -or mass- production then it uses half of human effort, half the manufacturing space, half of the investment tools and half of the engineering hours. As a result, it provides products with fewer defects and produces a variety of products because it works on the product line basis.

Lean software development mainly originated from the book "Lean Software Development: An Agile Toolkit for Software Development Managers" by Mary and Tom Poppendieck [3]. This book presents Lean production system with a new look for software development with a modified form of Lean principles including a set of tools.

2.2. Lean's Theory

A debatable issue in the software industry is whether Lean software development falls under the Agile development processes. Lean development process, inspired by the Toyota Production System (TPS), is more converging on some points strategically than the Agile process**,** while holding similarities to the process. According to Tomaszewski et al. [8], introducing Lean manufacturing idea for software development was not easy, because cutting the metal and make a car is much different from cutting the code and make a software with less cost. Development of software is rather different from handling operations and logistics.

When it comes to creating software, it is not just about "producing something"; it is the meaning of that "something" which should work to fulfill its purpose. The purpose is the satisfaction of the customer through the development process and with the outcome or product. In software development, adding value to the customer is equal to profit [3], so the equation of value calculation should look like below.

Value = Revenue – Expense
Revenue = Deported comprehended value
Expense = Development cost + Testing cost + Maintenance cost + Waste

Figure 2.1 – Equation of value calculation

When discussing software product value, cost and quality are the central looking points, affected mainly, according to Lean's theory, by waste. Lean, by design, focuses on identifying and eliminating waste [16] as fast as possible, and in turn, improving the software continuously by constant customer feedback. To avoid the non-value added activities, an organization needs to understand what value is and what resources are needed to create that value. It is true that no organization wants to create waste. When Lean is viewed from a software perspective, the Toyota example holds testimony to the advantages of Lean development. Lean comprises of principles that can be applied to improve the quality of a product in any given environment.

2.3. The 7 principles of Lean

According to Mary and Tom Poppendieck [3], there are seven main principles in Lean development process:

- Eliminate waste – Spending time on adding real customer value(s).
- Amplify learning – Increasing feedback to face tough problems.
- Decide as late as possible – Keeping options open as long as practical, but no longer.
- Deliver as fast as possible – Delivering value to customers as soon as they demand for it.
- Empower the team – Letting people who add value(s) to use their full potential.
- Build integrity in – Building product integrity into a system.
- Optimize the whole – Awareness to temptation to optimize parts at the expense of a whole system.

2.3.1. Eliminate Waste

One of the key principles that make Lean a successful development process is the elimination of waste. Lean development is, in principle, about reducing waste as much as possible, be it in a development team, group, or organization. Examples of waste include excess inventory, unnecessary efforts, duplicated data, and most importantly, cost related to all the aforementioned [16].

Software development is more of a tailor-made product development; it is not like duplicating the approved prototype. In software development organizations, the development process needs to be empirical. The reason for that is that software needs to adopt change, from its concept until its entire lifecycle. Eliminating waste is very important in order to reduce the cost and maintain the quality of a software product. To understand what waste is, organizations need to know what value is and which resources add value to the product. Mary and Tom Poppendieck [3] talked about waste related to software and showed the connection between software waste and manufacturing waste. Mark Windholtz also described waste in software development. The table below shows the software waste as described by Mark Windholtz and Mary and Tom Poppendieck [17].

Mary & Tom Poppendieck	Mark Windholtz
Waste of Software Development	
Extra features	Extra features
Extra processing steps	Partially finished work
Waiting including customers	Extra process steps
Defects not caught by the test or test failure	Waiting
	Defects
Finding Information	Motion
Requirements	Management Activities
Handoffs	

Table 2.1 – Waste of software development

Lean's value-adding activity is an activity that adds value to the product and process as defined by the end customer. Therefore, the non-value added tasks are simply the activities that the end user would not like to pay to perform and hence, wishes to wipe out.

In software development, elimination of waste can be coding activity by introducing the value added tasks and non-value added tasks also with reduction of the common errors. The concept of “Do it right the first time” is about not doing anything (or not start coding) unless one fully understands what the code is supposed to do and clear out all the requirements. Good understanding of the requirements and the domain, matched with short-build cycles and machine-driven testing is considered as the proper way of developing software.

2.3.2. Amplify Learning

Amplify learning is about planning to experiment, checking the results depending on the data, and then incorporating the things learnt. In software projects, this means deriving metrics that can be cross-team applicable, not just intra-team optimization. Organizations can do it with interconnected iterations across teams to increase inspection and adaptation [18].

Amplify learning specifically targets ‘Examine and Adjust’ from Agile practices. The need of this principle can be assured by realizing the problems that are barriers to success, but Lean software development in the planning phase identify the ways of solving those problems by amplify learning [18]. The importance of initial planning in Lean is essential for early requirement gathering and design specification with planning which can avoid finding information and motion waste; both of these two wastes can avoid planning to experiment with initial planning [19].

2.3.3. Decide as Late as Possible

A determination is a statistical reference and it is a hard decision to make during the development of software. This principle of Lean is based on the idea of making a decision at the last responsible moment or delay commitment. The main idea is to add value and avoid waste to maintain the cost and quality. In some other consequence, waste builds from extra features, extra testing, miss-concept architecture design and this increases the cost and decreases the quality of the product. The thought introduced here is pull, which was introduced by Mary Poppendieck. The idea of pull depends on the demand and downstream process required [20].

In a project, the requirements are assigned initially with some of them containing unsettled features. Therefore, it is hard to make the correct decisions until the uncertainty has been cleared or more information is made available from the people involved in the project. Lean development process supports delay in decision making for those uncertain tasks, keeping the focus on the currently running tasks and holding off as reasonable time as possible to implement the right product, aimed at limiting and reducing the waste. These principles mainly focus on avoiding the wasted extra features.

In Lean software development, in order to develop products with high quality and maximize the information flow for adding value, initial planning with shorter loops is used to make faster and cheaper products. On shorter loops, the requirement update can prioritize the requirements, with the high-priority ones to be implemented first, and also, test in the same loop can save time. So at the end of each feedback loop ongoing task is delivered. This is far more effective than a long loop. In increase to rapid delivery, Lean software development follows just in time information flow. One of the most in force carrying into action, Lean’s idea is delivering increments of business values in short time boxes [20].

2.3.4. Deliver as Fast as Possible

Delivering small pieces of software is easier than delivering a big product at once. Small software packages are easier to manage than a big one associated with fewer defects, obvious easier to integrate with an existing software system. According to Mary and Tom Poppendieck [3], one should limit tasks queue to minimum, with one or two iterations ahead. Items can be removed from the queue with Mary and Tom Poppendieck [3] claiming that if something important is removed, then it will not do any harm because customer will remind about it very soon. Important, valuable and urgent items need to be present in the queue (i.e. features that can add customer values). Teams need time to stabilize velocity and quantitatively see the deliveries at the end of iteration. Teams are expected to pull items from the queue based on measured velocity, and completely finish the work before they can move on. Moreover, developer(s) should reject any work until they have an empty slot in the queue. It is a tip because it does not make sense to add items to the backlog if they know they will not have time to implement them, unless they want to add something more important than the existing items [21].

2.3.5. Empower the Team

Empower the team consider with the decision making to the most depleted level primarily to those people who actually build the product and potentially add value to the release product. Empowering the development team to take part in technical decisions is fundamental to achieve excellence [3]. The main idea is to allow the people who have the most knowledge about a problem to make decisions on the way of solving the problem. Lean principles empower a development team to learn as they move forward in the development process and eliminate waste.

In Lean software development, the necessity of placing a high functioning team is important for success. As Lean is not just a control tool, it is an environment, an environment of a cultural movement with continuously improvement, which encloses improvement of human behavior and teamwork. In Lean software development, organizations that successfully implemented Lean, at the same time ventured an attempt to bring their individual employees together as a team and encouraged a team culture by rendering team coaching and facilitation aid [19]. In that way, an organization can construct high performance and crystal-line culture, where everybody can see the process movement and can satisfy the end customer, and also creates the understanding of optimization the software development.

2.3.6. Build Integrity In

Integrity (i.e. product integrity) has two dimensions: external integrity and internal integrity [3]. Internal integrity means that all the parts of a system work together. External integrity relates to the consistency between the performance of a system and customer demands. The principle supports the need for building product integrity with high quality and final integration defects avoidance, because product integrity and the essential resources to realize it can provide a sustainable competitive advantage to any organization. To understand the integrity of a product, an organization needs to integrate customer feedback into its development process. Therefore, product integrity can be achieved by the correct information flow and motion. That also dilutes insufficient information waste [22].

To maintain the quality of the software, defects should be inspected before the fact, have to control the condition of testing and integration. This principle is also known as ‘build quality in’, so the product should be inspected after each small step or loop. Agreeing with Shigeo Shingo, when a defect is discovered, consider fixing the defect first. In Lean software development, defect-tracking systems are queues that partially do the work. These queues are collection points of waste. The concept here is to keep the queue empty after each iteration, so at the final integration there will be no or less defects. In advanced software development, these can be done by test-driven development. This testing process, tests the system as frequently as possible, so the end product comes with built-in quality and integration [3] [4].

2.3.7. Optimize the Whole

Thinking about systems does exist, but the typical response to solve problems is to break them into their constituent parts and then optimize each individual part. This is sub-optimization, which leads to the “tragedy of the commons” and it does not work on optimizing the whole system. Optimize the whole value streams from the time it receives an order to address a customer’s need until software is deployed and the need is addressed, avoiding sub-optimization and encouraging improving a whole system, not just a part of the system [3]. Development teams also need to understand the problems of local optimization (i.e. local performance measurements in a department), which has a tendency to inhibit collaboration beyond the area being measured. Therefore, optimizing a whole system without sub-optimization or local optimization results in building the right product for the customer.

The goal of software development is to support the development of a complete product that fits the purpose of customers. Optimizing a system as a whole depends on customer collaboration, which is vital in Agile development, showing that the principle is related to Agile manifesto. A Lean organization optimizes the whole value from the time it receives the order to the time it delivers the right product to the customer.

3. Research Design

In this section, the design of the research project is described, along with reasoning behind it. Creswell [9] describes that a research design is mainly constituted from the research philosophy, the research approach (strategy of inquiry), the research method(s), and the data collection and analysis method(s). This section is structured in the same way – starting from the broader concept (the research philosophy), to the narrowest one (the specific data collection and analysis methods used in this study).

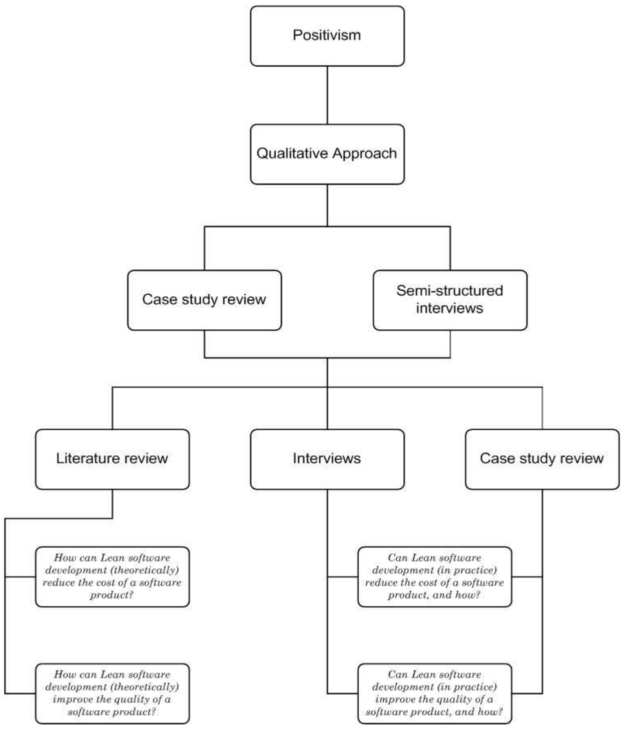
Positivism
Qualitative Approach
Case study review
Semi-structured interviews
Literature review
Interviews
Case study review
How can Lean software development (theoretically) reduce the cost of a software product?
How can Lean software development (theoretically) improve the quality of a software product?
Can Lean software development (in practice) reduce the cost of a software product, and how?
Can Lean software development (in practice) improve the quality of a software product, and how?

Figure 3.1 – Research design

3.1. Research Philosophy

The research philosophy is connected with the knowledge claims made within a study. Creswell [9] defines a knowledge claim as "certain assumptions about how they [researchers] will learn and what they will learn during the inquiry." The choice of a research philosophy mainly depends on the aim of the research project. As described above (see Section 1.2) the purpose of this study was to validate the applicability of Lean's theory in software development organizations. With this purpose in mind, the researcher reached in the conclusion that the predominant epistemological stance is positivist in nature. Positivist studies start with the test or verification of a theory. In positivism, theories are used in a deductive manner – they are found in the literature and then research is devised to test them. The positivist researcher "begins with a theory, and then collects data that either supports or refutes the theory." [9] The theory on which the research for this study was based is presented in Section 2.

3.2. Research Approach

As mentioned by Dalcher & Brodie [23], the research approach tends to follow from the research philosophy. Additionally, they describe that "the choice of research approach is strongly coupled to the type(s) of data available to the researcher." [24] However, in this case, there was a contradiction between these two. In literature, positivism is categorized as a quantitative approach [9] [23], but the data that the researcher aimed to collect in this study were qualitative in nature. That's owed to the fact that due to various limitations, the researcher was not able to conduct multiple case studies in organizations by himself. Therefore, the data were based on one case study of Lean implementation, and two interviews conducted within organizations that have implemented Lean (or similar methods inspired by Lean) as their development process. Hence, the study was based on qualitative data rather than quantitative as the research philosophy (positivism) suggests. Qualitative data consists of descriptions (descriptive data) and it is concerned with generating understanding and insight expressed in verbal descriptions [23]. The solution to this problem was found in the book "Case Study Research: Design and Methods" [24]. In this book, Yin [24] shows examples of how positivism can be used in qualitative studies. Hence, following Yin's [24] instructions, enabled the researcher to take a "positivistic" qualitative approach in examining the research problem.

3.3. Research Method

According to Creswell [9], the choice of a research method depends on the aim of the study. As described above (see Section 1.2), the aim of this study was to validate the applicability of Lean's theory, by examining examples of Lean implementations in various software development organizations. In literature, theory-validation studies are usually related with experiments or case studies. However, in the case of this study, by investigating only one organization which has implemented Lean would not have been sufficient, since there are many factors that could have affected the outcome of the implementation (the success rate can vary between organizations). Hence, the researcher had to investigate several organizations. Case study was selected as this study's research method because it allowed for an insight into organizations that are working according to Lean, in order to identify indications about the truth or validity of Lean's theory (by investigating the results and benefits these organizations gained). Additionally, it helped the researcher to develop an understanding on the implementation of Lean's theory in software development organizations, and to obtain deeper knowledge on the different steps and procedures involved within Lean implementation processes.

By having several organizations under study (large range of data) resulted in having more objective and reliable findings. The aim was to use these organizations as representative sample, with the intent of using the findings as "evidence" for arguing on the validity and applicability of Lean's theory in general. – "generalizing from a sample to a population." [9]

3.4. Data Collection

As mentioned above, this research was constituted from two phases: phase 1 (the theoretical phase), and phase 2 (the practical/empirical phase). During the theoretical phase, the researcher focused on collecting data regarding the cost- and quality-related principles/aspects of Lean's theory, while in the practical phase, data regarding the results – and possible benefits – of Lean's implementations were collected. The methods used for gathering the data during these phases are described below.

Literature review: During the first phase of this research (theoretical phase), a review of related books, conference papers and articles was conducted, in order to obtain a general understanding of Lean's theory and of its thinking principles. That also helped in narrowing the focus of the study and formulate the research questions. Based on the "gaps" that were identified in the existing body of knowledge around Lean software development (see Section 1.2), the researcher decided the focus of this study to be on validating Lean's theory in terms of cost-reduction and quality-improvement. In effect, keywords such as Lean software development, theory validation, reduce cost, improve quality, and Lean implementations were defined, in order to make an additional internet and library search. Reading the various abstracts and introductions of related research papers, permitted to make a collection of existing literature relevant to the topic of this study. The data that were gathered, allowed to identify and explore the theoretical assumptions of Lean related to cost-reduction and quality-improvement, and supported an answer to the first set of the research questions (see Section 1.3 – Phase 1 research questions). The findings established the groundwork of the data analysis, and are presented in Section 2.

Interviews: During the second phase of this research (practical/empirical phase), the majority of data was collected through interviews, taking the form of audio data as well as notes. Interviews were held with representatives from two companies (Ericsson AB, Tieto) which have been working according to Lean software development. The focus of the interviews was to understand why those organizations decided to implement Lean (or a process inspired by Lean) and what the outcome of this implementation was. That was done by exploring which characteristics of Lean "attracted" those organizations, what expectations they had in terms of results (before the implementation), and what were the actual benefits and results that they obtained (especially in relation to cost and quality). During the interviews, the questions asked were open-ended, in order to facilitate an open discussion. The data gathered, helped in validating if Lean's theory is successfully applied in practice, and the success rate of it.

Case study review: In order to systematically investigate and extend a practical industrial experience of Lean, one case study of an organization that is working according to Lean – Ericsson AB – was studied. This case study is an early evaluation of the implementation of Ericsson's development process called Streamline Development [8], which is based on the thinking principles of Lean software development. By examining the case study from Ericsson, the researcher developed an understanding of the different steps and procedures involved within Lean implementation efforts. Additionally, this was used (together with the interview results) in order to compare with the theoretical findings from phase 1. Through the case study review (as well as the interviews), the researcher was able to look inside the use of Lean in various organizations and accumulate evidence for supporting his statements, from the tried examples of Lean implementations. The data gathered from phase 2 was used to answer the second set of research questions (see Section 1.3 – Phase 2 research questions).

3.5. Data Analysis

After collecting the data, their analysis was performed using content analysis. Content analysis is an in-depth analysis using quantitative or qualitative techniques of messages, and is not limited to the types of variables that may be measured or the context in which the messages are created or presented [25]. In content analysis, the analysis of the data is being accomplished by reducing them into thematic categories. In this study, the data was divided in two thematic categories: cost and quality. The division of data into categories explores the relationship between the concepts (themes) identified.

Throughout the data analysis, the researcher looked for specific words like time, cost and quality. The analysis was conducted in two phases. During phase 1 (theoretical phase), the focus was on analyzing the data collected through the literature review, providing answer to the first set of research questions (see Section 1.3). In phase 2 (practical phase), the empirical data from the interviews and the case study review were analyzed, where, in comparison with the findings from phase 1, enabled a discussion around the validity of Lean's theory, providing an answer to the second set of research questions (see Section 1.3).

4. Results

The aim of this chapter is to present the findings of the practical/empirical data collection phase, along with descriptions about the organizations under study and their development processes. The process descriptions presented in the following sections are simplified since the main focus of this study is on the results and benefits that those organizations gained, rather than on their processes, hence the intention is to give the reader a general understanding of those processes rather than describe them in detail.

4.1. Interview 1: Ericsson AB

The first interview was conducted at Ericsson's site in Göteborg. Ericsson is one of the major software developers of telecommunication systems in the world. The organization representative (interviewee) was a project manager from that site, who was also involved in the implementation of the company's new development process (Streamline Development). Streamline Development (SD) is a custom process developed by and for Ericsson AB, which is based on the thinking principles of Lean software development. As stated in the interview "Streamline Development is a specialized instance of Lean development." The main goals of SD are to improve customer responsiveness, identify and eliminate waste, optimize the whole, and increase flexibility.

Prior to SD, Ericsson was using another company-tailored process that was similar to Rational Unified Process (RUP). However, the problem with that traditional process was the long time-to-market – due to the long life cycles of the projects (often more than a year). "Traditional processes tend to have rather long life cycles and do not deliver the actual customer value [i.e.

the working system] until late in the process." [26] Another problem was flexibility (coping with changing requirements/customer needs) – due to the long duration of the projects, the company was especially exposed to changing market – and customer – demands. In order to overcome the problems of their traditional development process, Ericsson developed a new in-house approach, tailored to the specific characteristics of the company.

Streamline Development is an incremental process that focuses on customer responsiveness and elimination of waste. In SD, the projects are significantly shorter (around 3 months lead-time) than in the traditional process described above. This result in delivering the products to the customers more quickly – the system (or part of the system) is available early in the development process. The size of the projects is also smaller in SD (significantly fewer project members). This means that the scope of the projects is also reduced [8].

Another aspect of SD is version integration. In SD, new system versions are integrated with the current product baseline. Hence, there is always only one version of the product at any point of time. It should be noted that even though each project produces a new system version that potentially can be released, it does not have to be released to the market. Such a separation between development and release is a clear difference from the traditional model, where each project ended with a release of a new version of the system to the market [8].

Also, another important characteristic of SD is continuous requirements prioritization. In SD, all the requirements of a product are gathered in a requirement repository, where they are categorized/prioritized based on their importance (high to low). When there are a suitable number of highly prioritized requirements that can be combined into a requirements package (based on that they fit well together, etc.), a new project is initiated. Continuous requirements prioritization of the requirements in the repository assure that only the most "pressing" (highly demanded) requirements are implemented for each new release of the system. The size of the requirements package is limited by the project length – it should be possible to implement the requirements from the package within the project boundaries, i.e. in about 3 months [8].

As noted within the interview, Ericsson wanted to get shorter time to market and also flexibility with the customer, and that was the main idea behind the new process and the change. Traditional processes are more expensive to be flexible than Agile. Main problem with the time, it cost too much for them to get flexible. Traditional development processes are more expensive and take more time to deliver the product when it is flexible.

Ericsson liked some interesting aspects of Lean. From Ericsson's standpoint, Lean has different ways of looking waste; focus on the main activity and what feature need to deliver fast. Traditional development processes have some unnecessary steps which increase cost and timeline of the product and the customer gets unhappy with the result. But on the other hand, Lean software development helps to deliver the features depending on the customer demand. Lean's theory believes that building something that is not used right now is the worst thing.

Ericsson was looking for new concept to find their waste and a proper optimization from start to the end of a project. Customer benefits were the main goal for the company. The main aspects of Streamline Development were eliminating waste and optimize the whole. The concept of looking the waste was interesting. Basically there is no difference on the foundation and in terms of goals. But when an organization or company have their own process then they have better control on it, and also they can adapt changes. Streamline Development is one specialized instance of Lean development.

It is mainly the motivation which may differ from Lean software development. There is no difference in terms of goals. The platform is the same; the only difference is in the name - in order to adapt changes and to increase control on the process. It is a continuous development process. Renaming it to Streamline Development helped Ericsson to adapt the changes by continuous improvement. SD is their own way of working depending on their own demand. Ericsson mainly focuses on customer demand, flexibility, maintenance time and market demand, in order to release a new product to reach lower lead-time to customer. The main intensions of SD were to increase flexibility regarding customer needs and demands, improve the quality and address time-to-market, and lowering the time with higher flexibility.

In order to avoid sub-optimization, Ericsson tried to make the goal clear by optimizing the whole - increased understanding between different units. Ericsson is a cross-organization, so optimization and communication between them is important. In terms of expectations, the new process assisted them a lot. Especially in terms of times and flexibility, it was proven that Ericsson met their expectations.

Ericsson thinks that Software Development is a continuous improvement process. They expect to see more and more of the benefits that they have so far. Lean development is an ongoing process of improvement, so each time Ericsson removes a bottleneck they find, the same area can be updated in the future. It takes time to understand the process on practical level. It needs attention and more focus to get improved.

The more "wastes" they find, the more benefits they gain. After using it for some time, Ericsson obtained some benefits, but the real benefits are hard to measure now because it takes time to understand the process. Ericsson wants to get more improvements which, by removing the bottlenecks, they can make the process better and gain more benefits.

After introducing SD, Ericsson has decreased development costs, but it is hard for them to measure development cost because there is no method for measuring cost-efficiency. It's easier to measure in manufacturing, but not in software engineering. In science, there is no known method to measure the cost for the development of software, but as they said, their productivity has increased a lot.

Lean principles also helped them to identify bottlenecks, for example, the value-adding activities. It is the way to see what the customer wants. For example, one waste can be keeping track of the tasks, or a to-do list queuing up a lot of things. Ericsson minimized waste by less paper work, not much market analysis, and faster flow with fewer things to look. Lean reduces the managers' work and makes it simple for everybody, because now the managers have to keep track of fewer people. By implementing less waste, Ericsson increased the quality of their products.

In Ericsson, the scope of the responsibilities has increased after introducing SD, since teams have now more "organization-oriented" or "overall process-oriented" responsibilities, instead of "specific process-oriented" (ex. testing, developing, etc). Now the employees care about the whole process, not just their part.

4.2. Interview 2: Tieto

The second interview was carried with Tieto. Tieto is a consulting company working with IT, R&D and consulting services. With approximately 16300 employees, Tieto is one of the leading IT service companies in North Europe and became the global leader in selective segments.

Tieto focuses in areas where they cover the deepest understanding of their customers, businesses and needs. Tieto is working with different types of organizations in Northern Europe, Germany and Russia. They serve their customers globally in different sectors like telecom, forest, oil and gas, as well as digital services. Tieto was a Finish company founded in 1968, Enator was a Swedish company founded in 1995 and TietoEnator was formed by the combination of Tieto from Finland and Enator from Sweden in 1999. On 26th March 2009 it becomes Tieto Corporation. The organization representatives (interviewees) were two project managers from Tieto, who were also involved in the implementation of the company's new development process called D2M (Design-to-Market). D2M is a combination of 12 Agile principles; it's a process to ameliorate man to man communication. D2M is a process which overcome difficulties normally faced by Tieto. D2M avoids waste and improves the end product by adding customer value. D2M works in smaller iterations in order to avoid miscommunication. Additionally, the process contains self-improvement.

In Tieto, the process they used in early 1990's was called PPS/PPM (Practical Process Management). Tieto was using that process for a long time. It's a well known process in Sweden and Scandinavia, and PPS is a process developed by Enator in 1990 or even in 1980. It's close to RUP (Rational Unified Process). Tieto has a policy that if the customer asks them to use Agile or some other new development process, then they have to adopt Agile. PPS/PPM was very common a few years back, but today it depends on the project managers. They can use whatever they want, and almost all project run with modern development processes like Scrum, Agile or other similar type of processes. PPS/PPM is a process which was developed and maintained to make software in 1980's. It can be productive and effective like the modern ones, but it depends what you want to deal with it.

By interviewing two project managers of Tieto, they explained/ stated that they have good experience working with Scrum. In their words: "Actually the sprints are which makes Scrum good. The sprints and the short increments result in much quicker development if you compare with any other development process." Tieto prefers to keep it simple. They think configuration management is very enormous and important for any process model .So by having smaller increments, it's better to avoid miscommunication and information with less fixing.

Tieto tried to implement 12 Agile principles and it was difficult to follow, so they tried to come up with a process to ameliorate man to man communication. The intention was if they run a project between different sites, then they have lots of miscommunication, and miscommunication creates misconception about the project, which results in the end product to become a fail. So they tried to implement a process which will overcome all the difficulties they normally face with their development process. D2M (Design to Market) was planned to avoid waste and improve the end product by adding customer value. It has smaller iterations which help in avoiding miscommunication and deliver the product faster to save time with more superiority.

D2M is an iterative development process with five phases and eighteen sub-phases. It has Analysis, Design, Implementation, Test, and Advance as its main phases, and as sub-phases it has Requirement Analysis, Architecture Analysis, and Object Analysis (inside Analysis phase), Architecture Design, Object Design, Impact Diagnose, and Test Design (inside Design phase), Test Implementation and Unit Implementation with unit testing and check and merge together (inside Implementation phase), Automated Testing and Evolution, Prototype (inside Test phase), and Custom Evolution, Code review, Re-factory and process improvement (inside Advance phase).

The inspiration of D2M came from many new development processes. It combines parts from Lean, Agile, and other similar modern development processes. Tieto tried to pick up the main idea or common sense from these processes and keep the aspects they liked. D2M is not a shadow of Lean, but in some sense it's kind of alike to Lean but some of its aspects they are not. If you run D2M, you will notice influence causing something without any direct or apparent effort, because it has resemblances. Actually it's a new process based on Lean and Agile.

As Tieto have the Agile manifesto, then it's normal that D2M is kind of analogous to Lean because in some point they are similar. Conceive about the principles, they have similarities in Eliminate waste, Amplify learning, Decide as late as possible, Deliver as fast as possible, Build integrity in, Optimizing the whole, and Empower the team.

Comparing to Lean or any other Agile process, what D2M attempts to avoid is the big iterations. If Tieto worked with four- or six-month iterations and after the iteration they found some bugs or a crush, then it would be a big waste and also lots of work to do for them. Additionally, the task of finding the bugs (debugging) would become a lot more difficult.

As Tieto explained, if they work in a big iteration for four months, after that, when they check the code they will find lots of problems with the main line and that will take them at least two working days to fix the problems and start again. But if they work in small iterations (like two to four weeks), if the code crushes then it will take 5-10 minutes to fix the problem, and on the second iteration will also take similar time to fix. So with small iterations it will take 20 – 40 minutes to fix the code instead of two days work. They have an auto-build process which checks the code every night so they know the fault when it's still "fresh", not after four months.

So what they intend to do by testing the code as soon as possible, is to reduce the time that takes to fix the code and minimize the cost for testing the code. Usually if they detect the problem faster, then they can also fix it faster. It's better to fix it when it originates rather than leave it for later. Even if they have the impression that they can fix it a lot faster than two days, after four months they will probably not remember what to do.

The motive for introducing D2M was mainly the quality and cost. Tieto wants to gain lower cost to market by doing the right things on the right order. So the new process is about saving money and time by maintaining the quality. However, the quality depends on what kind of quality attributes the customers want to focus on, because there are too many quality attributes they can focus on.

The problems Tieto tried to solve are on a theoretical level, but for the most part, D2M focuses on the cost and quality attributes that Tieto want to reach from the beginning. For example, continuous integration helps to solve lots of problems. After encountering a problem, they focus on fixing the problem first, so then they can say "it's a success".

D2M is a combination of many infrastructures to reach goals with high quality and low cost. It is not only just the Lean principles; its lots of other manifestos from Agile combined with Tieto's own view of looking the project.

In D2M (which is partially inspired by Lean), after each iteration, the process has an improvement phase. On this phase, Tieto tries to identify the bottlenecks and update the process, so the next iterations are safe from similar problems.

D2M helped Tieto to identify and remove non-value adding activities, for example, if developers have implemented some features which are not necessary or if they have moved necessary features aside and instead, implemented some unnecessary ones. This is something they would usually notice at end of the iteration with the help of the customer. So if they have done something which is not important from customer's point of view, then it's an overwork. However, in their case they don't call it waste; they call it misunderstanding between the company and the customer, but in Lean it's considered as waste - you worked more that you should. Waste for them is more than this, for example, what they mentioned during the interview "If someone (individual worker) disappears for three days and after that he/she comes back with something which is not working, that's a waste" (waste of time or in other case can be waste of quality or money). In their working process, they have the daily checks which are very important for them since they can monitor everything, and if they can successfully do that, then they can just avoid the misunderstandings or problems of extra working easily. So daily progress checks actually helps Tieto a lot. This way they also measure progress. Additionally, the process helps them without sourcing since they have time for check and merge in implementation phase and they have a phase called 'Advance'. Inside this phase, they do custom evolution, code review and re-factory, which help them reducing a big amount of waste. So by estimating waste efforts that have any impact on cost or quality was really small for the company.

The principles of Lean helped Tieto to lower development cost. In fact, they have lots of automated code and a build program which is also automated. "Auto-generated code is normally faster and automated build also save lots of time and money." Auto-generated coding, testing and building work together with continuous integration. If they didn't have continuous integration, then they would have to employ another person to do it. So that's also less extent cost and everything is handled by the process.

When the researcher talked about higher quality, the interviewees explained about quality control and how the new process helped the organization to obtain high quality. What they said is that the process extents their capability to control and manage the positive qualities, especially those suitable for specific customers - they have much better quality control in this process. They actually grantee that what they deliver is higher quality products; they don't have any shortcuts in their code. In a long run, they believe that it's a good investment. It seems promising - the projects run in a faster way and with lower cost while, at the same time, the quality is secured. Tieto and its new process focus on quality, cost and times as their best cognition.

As a consulting company, Tieto always aimed for customer value, thus it's very important for them. With the right product on the right time, they also provide additional services, for example, they provide support, they have installation, etc. But that's what they try to do as an extend. That is why they are having morning meeting; in order to provide this great additional value to their customers. "It can't be measured with money, because we try to satisfy our customers to make them come back." Relationship with the customer is valuable for Tieto. In their process (D2M) they have an extra layer which adds additional value to customers, which is very important for them to do business.

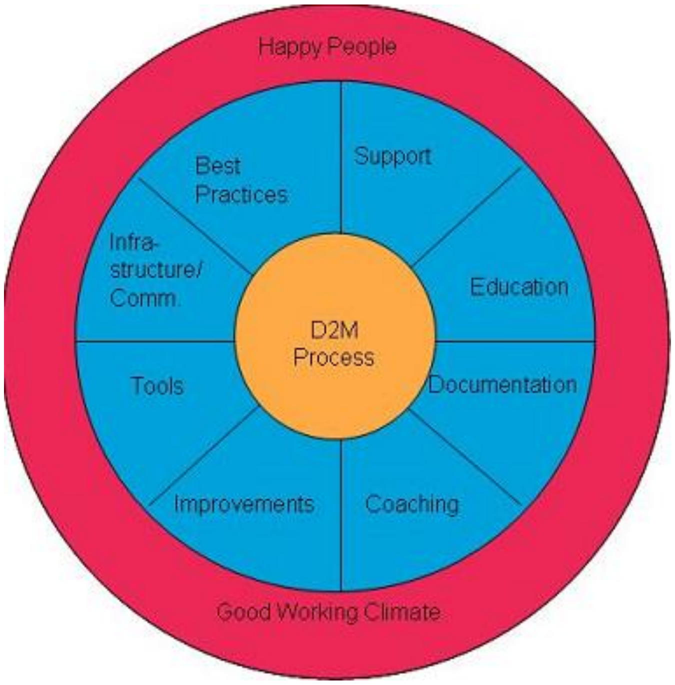

Figure 4.1 – The whole product

Tieto thinks after presenting D2M that in some cases the responsibilities for individuals have increased, but at the same time, they also have the motive that they don't need to do the same work twice. So in a sense, the amount of work was decreased.

4.3. Case study review

Apart from the interviews, a case study review was also conducted in order to gather more "concrete" data. The reviewed article is "From Traditional to Streamline Development – Opportunities and Challenges" by Piotr Tomaszewski, Patrik Berander, and Lars-Ola Damm [8]. The article presents an early evaluation of the suitability of Streamline Development for Ericsson (Section 4.1 for information concerning SD). The evaluation was performed by finding positive and negative aspects (benefits and drawbacks) of introducing SD, as well as identifying changes required to prepare the organization and its products for successful implementation of SD. The goal of that study was to provide decision makers at Ericsson with a deeper and more structured understanding of the possible effects of introducing SD. The study was performed in two product development units (PDUs) at Ericsson, and the data regarding the impact of introducing SD was collected in a series of interviews with representatives of the roles in the company that would be most affected by changing the development process. To analyze the findings from the interviews, a modification of Force Field Analysis (FFA) was used by the researchers. "FFA is a method for identifying issues that should be taken into account when deciding whether to implement a strategic change." [27] The findings were then structured and categorized by the researchers into three categories: Pushing factors, Resisting factors, and Required changes. Pushing factors were regarded as the advantages of SD (things that would improve after introducing SD), while Resisting factors were regarded as threats connected with introducing SD (i.e. things that would worsen by introducing SD). Required changes were regarded as issues that should be resolved and problems that should be overcome before SD could be implemented. By balancing the Pushing and the Resisting factors, it was possible for the researchers to make an informed decision if the change was worth introducing or not. The information about Required changes made it possible for decision makers at Ericsson to assess the cost of introducing the new process (SD) and to identify issues that should be resolved before SD could be introduced. A detailed description, together with an analysis, of the findings mentioned above is given in Section 5.

5. Analysis & Discussion

In this section, the data of this study are being analyzed, and the results of the analysis are discussed in detail. For the purpose of this analysis, a comparison between the two aforementioned sets of data is conducted. First, the theoretical principles of Lean related to cost-reduction and quality-improvement (identified in phase 1) are presented, together with a description for each principle and for how it is argued by the theory that it can help in reducing the cost and/or in improving the quality of a software product. This set the theoretical ground of the analysis. Then, the empirical data from phase 2 (case study review and interviews) are presented. Those data represent the application of the theoretical principles mentioned above in two organizations (Ericsson AB, Tieto), and the results that were obtained. That allowed an investigation on how those companies made use of the theoretical principles, what results they obtained by using them, and the success rate of that (if the obtained results reach their expectations). These data are then used as an analysis/validation tool, in order to support or refute the validity of the theoretical principles of Lean (from a cost and quality perspective). The comparison between those two sets of data (theoretical and practical) facilitates a discussion on the validity of Lean's theory, which results in verifying if Lean achieves in practice the goals it was designed to achieve (e.g. reduced product cost and increased quality).

This section is structured as follows: Section 5.1 analyzes and discusses the principles/aspects of Lean associated with cost-reduction, while in Section 5.2 the quality-related principles of Lean are being discussed.

5.1. Cost-reduction

The most cost-related principle of Lean is *Eliminate Waste*. This principle focuses on reducing the development timeline by removing all nonvalue-adding activities (create nothing but value). Value is giving the customers what they consider as important, at the time and place where it will provide the most value. Hence, anything that does not add value to a product is waste, and any delay that keeps the value from the customer is waste.

The first step to eliminate waste is to recognize it. In order to do that, an organization must first determine what value is. In other words, to develop an understanding of what customers really want – what they will actually value once they start using the product. That's different for each customer, and can vary between many different things (quality, performance, cost, etc.). Once the organization develops a keen sense of what value really means to them and their customers, then they must develop a capability to really see waste. In other words, to identify the activities that interfere with delivering customer value (nonvalue-adding activities), and eliminate them. "The ideal is to find out what a customer wants, and then make or develop it and deliver exactly what they want, virtually immediately. Whatever gets in the way of rapidly satisfying a customer need is waste." [3]

Mary and Tom Poppendieck [4] identified 7 wastes of software development (for details, see Section 2.3.1). Out of these, the ones most related to cost are:

- Partially done work
- Changing requirements
- Extra features

The above "wastes" play an important role in increasing the cost of a software product. Firstly, by adding additional time (i.e. time to find and fix defects, time to implement unnecessary functionality, etc.) and, as being known, time equals money. Also, by creating complexity (which is a large cost multiplier). Complexity makes the code base brittle and difficult for future changes. "The cost of complexity in code dominates all other costs, and extra features that turn out to be unnecessary are one of the biggest killers of software productivity." [4] Therefore, Lean's theory believes that by identifying and eliminating counter-productive activities/processes which do not add any actual value to the customer and the product, can effectively reduce the cost of a software system.

In both organizations under study (Ericsson, Tieto), the adoption of a "lean way of thinking" helped them in identifying numerous wastes associated to their previous work practices. In particular, in the case of Ericsson, one of the main problems in relation to their previous development process (see Section 4.1 for information on Ericsson's traditional process) was the high cost for maintaining different product branches. In Ericsson, when a product is developed aiming for a broad market launch, it is common that specific customers ask for adaptations which should not be included in the main release, and hence, in the main project (customer adaptation projects). In the past, when a customer adaptation project was initiated, the requested functionality was implemented into a branch of the system. Upon completion of such a project, the product was released to the ordering customer but was not integrated with the main product, and hence, had to be maintained as a separate product branch. However, maintaining multiple branches/ versions of the same product was very costly for Ericsson. "Each such adaptation [project] has to be maintained, almost as a separate product, which makes them very expensive." [8] When the company adopted a "Lean" way of thinking and looking at waste, it enabled them to identify this drawback, and address it in their custom (Lean-inspired) development process called Streamline Development (see Section 4.1 for a description of SD). In SD, customer adaptation projects are handled in the same way as any other project, i.e. they are integrated into the main product. This means that only one latest system version needs to be maintained. Hence, maintenance is simplified and cheaper. "In SD there is only one version of the system produced, with no branching for customer adaptations. That minimizes the number of maintained system versions and, therefore, minimizes the cost of maintenance." [8]

Another waste identified by Ericsson, was connected with excessive documentation during projects. The same waste was also identified in the case of Tieto (see Section 4.2 for information on Tieto's traditional process). Excessive, unnecessary documentation is a common waste in traditional development. In both Ericsson's and Tieto's traditional projects, a considerable amount of time was spent for producing "routine" paperwork. Most of these documents had no other use than just fulfilling "archive" purposes (hence, they were adding no actual value to the product). Nevertheless, they still had to be produced since they were defined as "standard" by the process. The downside of that however, was to increase the workload of the people working on the project which, in turn, had a negative impact on flow. By flow is meant the flow of information and delivered value. As described by Poppendieck [20], handing off reams of frozen documentation from one function to the next is a mass-production mentality. In Lean, the idea is to eliminate as many documents and handoffs as possible. Documents which are not useful to the customer are replaced with activities that provide customer value. Hence, the minimization of flow was affecting the productivity of both organizations in a negative way. Additionally, increase of the workload was resulting in overall project's timeline increase (time-to-market) – since the project team had more tasks to take care of. "When you overload something, delay increases." (Project manager – Ericsson) And when delay increases, cost is also increased. When both organizations started looking at things (and in particular, the process) from a "Lean" perspective, it enabled them to identify unnecessary steps which were interfering with providing value to the system. Once these nonvalue-adding activities got eliminated and the focus was steered towards the ones that created direct value for the customer, dramatic improvements were obtained. "The more "wastes" we find the more benefits we gain." (Project manager – Ericsson) In fact, by removing unnecessary documentation, the workload was reduced, resulting in increased flow, and hence, productivity. Moreover, the development timeline was minimized, which, in turn, had a positive impact on cost (less development time = less [development] costs).

In summary, Lean software development helped both Ericsson and Tieto to identify and eliminate several activities which were keeping the value from being delivered to the customer and were adding unnecessary costs to the product. Hence, the adoption of Lean principles (and most importantly, of the principle *Eliminate Waste*) enabled both organizations to get rid of such non-value, time-consuming processes, and resulted in reducing the cost of their software products.

Another cost-related principle of Lean is *Deliver Fast*. This principle put emphasis on delivering increments of real business value in short time-boxes. On the other side, one of the main drawbacks in relation to traditional way of working is the long cycle times (i.e. time-to-customer). "Traditional processes tend to have rather long life cycles and do not deliver the actual customer value [i.e. the working system] until late in the process." [26] Late delivery of value results in reduced customer responsiveness and feedback. Additionally, traditional processes are not very good at coping with changing requirements, since they are especially exposed to changing market demands due to their long duration.

In a technology and market-driven environment such as software domain, it is common the market and its customers to change their minds while the project is still ongoing. This leads to a situation where some of the already specified and worked with requirements become obsolete. In comparison to the initial set of requirements, some requirements need to be added, some need to be changed, and some need to be deleted to better match the customer expectations. However, as time moves forward in a project, modifying the system tends to get more difficult and expensive (the later a change is identified, the harder and more costly it gets to address it). Changes add complexity – which, as described above, is a large cost multiplier – and complexity usually calcifies the code base, making it brittle and easy to break. Furthermore, dealing with changes of already decided and sometimes implemented functionality, decreases productivity and increases project lead-time (and hence, cost). "Changing requirements are a source of significant amount of rework to adapt the system to new requirements [customer needs], and such rework is one of the most important productivity bottlenecks in large projects." [28]

Lean's theory argues that the risk of rework connected with changing requirements can be minimized by shortening the development cycle and involving customers early in the process. In Lean, the system is divided and developed in small, rapid increments driven by customer priority and feedback, instead of one, long iteration where the customer gets involved towards the end (as in traditional development). Hence, Lean organizations focus on cycle time, not utilization. As described by Poppendieck [4], companies that compete on the basis of time often have a significant cost advantage over their competitors, since they have eliminated a huge amount of waste (which is the extra time). Furthermore, they have better chances for self-improvement, since they are so fast that can afford to take an experimental approach to product development, trying new ideas and learning what works. In software development, the discovery cycle is critical for learning (design, implement, feedback, improve). The shorter these cycles are the more that can be learned [3]. Rapid development also assures that the customers get what they need now, not what they needed yesterday (up-to-date functionality). Additionally, it "deludes" them into delaying making up their minds about what they want, until they know more – since it will be easier for them to decide what they actually need (or not need) once they get involved in the process, than by just being "observers". "Figure out how to deliver software so fast that the customers don't have time to change their minds." [4]

According to Lean's theory, another benefit of rapid development is improved productivity. Shorter projects tend to have more stable scope because they are less exposed to the risk of changing market demands. Stable scope minimizes waste since not that much rework must be done to adapt the system to new requirements, and therefore, leads to higher productivity.

Summarizing, in Lean projects, the system (or part of the system) is usually available early in the development process. This makes it possible to meet customer needs faster and deliver value to the customers earlier. Frequent releases enable early customer feedback, which results in getting the customers more involved in the entire process. This, in turn, makes possible to detect and address changing needs much earlier, and thus, improves customer responsiveness and reduces the risk of waste (and extra cost) caused by implementing inadequate functionality.

In both organizations under study, one of the main problems in relation to their previous work practices was the increased number of changing requirements within their projects and the high cost for coping with them. As described above, that was mainly owed to the rigidity of these practices, due to the extensive length of their iterations – both companies stated that the duration of their traditional projects was very long (in some cases even years). "Traditional development processes have some unnecessary steps which increase the cost and timeline of the product, resulting in the customer being unhappy with the result." (Project manager – Tieto)

In the case of Ericsson, prior to the adoption of a "Lean" way of working, the company was developing systems in a rather traditional style – in large projects which had long life cycles (often more than a year). However, the long project lead-times decreased the company's competitiveness by lowering customer responsiveness, since as stated by Tomaszewski [8], it is always harder to make a change of something that already is implemented than of something that is not yet specified in detail. In Ericsson's traditional process (RUP), changing requirements were addressed in Change Requests (CR). CRs were lists of the changes that customers demanded every time they were identifying a change in their needs. These changes usually involved addition of new requirements, and/or modification or replacement of some of the already specified ones. In other words, when the customers were changing their minds about something (usually due to changes in market demands), they were addressing these changes in a Change Request. However, due to lack of customer collaboration in RUP projects, the customers were usually identifying and requesting these changes late in the process – where most of the initial requirements had already started (or even finished) being implemented. Hence, when a Change Request entered a project, it often implied that an "original" requirement which was already implemented should be either re-implemented or thrown away depending on the kind of CR. In either case, already performed work was thrown away, which means that CRs entering a project led to some kind of waste. "Change Requests identify and is the trigger of waste and rework." [8] Hence, the high cost of handling CRs was one of the main problems Ericsson had with their traditional way of developing software (RUP).

When Ericsson adopted Lean thinking – by implementing SD – the length of their projects was reduced (from 1 year to 3 months), since one of the main characteristics of Lean development is delivering the software in small increments. According to Tomaszewski [8], the most evident difference between SD and RUP is much shorter time between identifying and implementing needs in products. In SD, the projects are significantly shorter than in the traditional process (RUP), and the size of the projects is also smaller (significantly fewer project members). This means that the scope of the projects is also reduced. Due to smaller project size and scope, SD improved the controllability of Ericsson's projects; the reduced size of the projects enabled the organization to make more correct predictions and estimations regarding the projects' lead-time and cost. Small projects also made easier for them to obtain and maintain an overall picture of what is happening within the project and, therefore, to monitor progress more easily.

In addition, as SD-projects are much shorter than the traditional ones, they tend to have more stable scope since they are less exposed to the risk of changing requirements – the change of market demands in 3 months is considerably less probable than their change in 1 year. As described above, change in requirements commonly involves waste because already performed work has to be remade (or removed) to adapt the system to the new requirements. In Ericsson, by improving the stability of their projects' scope (through the reduction of their timeline), resulted in minimizing the risk of waste (unnecessary work and cost) from changes in requirements, which in turn, led to higher productivity.

Similar benefits were also identified in Tieto's case. The company's Lean and Agile-inspired process (Design-to-Market) also works in small iterations (2-4 weeks). "What we are attempting to avoid is the big iterations. If we work with four or six month iterations and after the iteration we find some bugs or a crush, this is going to be a big waste and also lots of work to do [rework]." (Project manager – Tieto) As further stated by the interviewees, reducing the development life-cycle (shorter increments) was proven very beneficial for the organization. In particular, Design-to-Market (D2M) gives Tieto a competitive advantage by increasing customer responsiveness, since the products are being delivered to their customers faster. The short time between ordering functionality and getting it is very attractive from the company's customers' perspective. The customers are interested in getting new systems with new features fast, since these systems often give them a competitive advantage. Another characteristic of D2M in relation to rapid development and delivery is the ability to release more often than Tieto's previous (traditional) process. By releasing new system versions more frequently, resulted in increased customer collaboration and feedback which, in turn, gave the ability to the organization to quickly respond to new customer demands.

Concluding from above, the use of the *Deliver Fast* principle had positive effects in both organizations which were interviewed. In their own words: "Lean helps to develop and deliver faster to the customer." (Project manager – Ericsson); “Short increments result in much quicker development, if you compare to any other development process.” (Project manager – Tieto) In fact, the most “profitable” benefits that both companies obtained are:

- **Improved customer responsiveness:** Rapid feedback on the correctness of estimations; end-product closer to customer expectations.
- **Minimized risk of changing requirements** and improved ability for effectively coping with them – due to increased customer responsiveness).
- **Reduced cost of managing Change Requests:** The cost connected with managing CRs was reduced, mainly because the number of CRs was decreased due to shorter projects.

Hence, developing and delivering as fast as possible helped both Ericsson and Tieto to reduce the cost of their software products.

As described above, owing to the long lead-times of traditional projects, changes in requirements are relatively common in traditional development and account for a significant part of project cost. However, long iterations are not the only reason why such practices are unable to effectively cope with changing customer needs. Another drawback identified in relation to the traditional way of working is early requirement specification and commitment. In traditional processes, starting development with a complete specification is considered a good practice. This type of development falls under the classification of the deterministic school of thought. As described by Poppendieck [4], the deterministic school starts by creating a complete product definition and then creates a realization of that definition. In other words, the deterministic way of working is: Make a complete product definition from the beginning, then create a plan and stick to it (commit). However, in an evolving environment such as software domain, changes in customer and market needs are common. Hence, when requirements are specified long before coding, there is a high risk that they will change along the way. Additionally, when deciding upon everything from the beginning, the design is partially based on forecast – since some things are unknown (or, at least, not clear enough) at the beginning of a project. Thus, early requirement specification and commitment increases the risk of changes in requirements and of the waste (rework/cost) associated with them.

In Lean’s theory, this issue is addressed in the principle *Decide as Late as Possible*. The essence of that principle is that in uncertain situations (such as in the beginning of a project), critical decisions must be delayed, while the focus should be on maintaining options. In such situa-tions, delaying decisions is valuable because better deci-sions can be made when they are based on fact, not speculation. “Development practices that provide for late decision-making are effective in domains that involve uncertainty [such as software domain], because they provide an options-based approach.” [3] Additionally, in order for organizations to effectively cope with changing requirements, they should build a capacity for change into their systems. One way to achieve that, is by trying to make most of the decisions reversible, so they can be made and then easily changed – reversible decisions are easier to change when it is required (they can better adapt to changes). Moreover, in the cases where an irreversible decision must be made, then – according to Lean’s theory – it should be scheduled for the last responsible moment – that is, the last chance to make the decision before it is too late. In conclusion, by having change in mind when making decisions and commitments is a very effective strategy for dealing with unexpected changes, because it adds flexibility to the system. “While we are developing the early features of a system, we should avoid making decisions that will lock in a critical design decision that will be difficult to change.” [4]

Another way for building change tolerance into a system is by maintaining options at the points where change is likely to occur. The best way to achieve that is by making plans. Within software projects, creating several (alternative) plans is particularly effective in situations where tough problems need to be tackled, because it enables the development team to experiment with various solutions. Another reason why planning is useful in uncertain, complex situations (such as the initial phases of a project), is because it leaves critical options open until a decision must be made. As described by Poppendieck [4], planning is an important learning exercise, it is critical in developing the right reflexes in an organization, and it is necessary for establishing the high-level architectural design of a complex system. However, it should be noted that plans should aim at creating options (alternatives), not commitment. “…orders have to change rapidly in response to change in circumstances. If one stick to the idea that once set, a plan should not be changed, a business cannot exist for long.” [2] Hence, keeping options open (by means of plans) is more valuable than committing early (making a plan from the beginning of a project and “stick” to it until the end), because it improves the flexibility of an organization and their ability to respond to changing customer needs (changes in requirements).

Summarizing, by making reversible decisions and maintaining options, builds a capacity for change into the software system. That combined with short development cycles and increased customer collaboration and feedback, results - based on Lean’s theory - in minimizing the risk and significantly reducing the cost of one of the biggest drawbacks in traditional development: changing requirements.

Nevertheless, changing requirements are not the only big problem of traditional practices. As noted by Mary and Tom Poppendieck [4], the worst (and most costly) type of waste in software development is extra features. Extra features are secondary, unused features that do not provi-de any actual value to the customer/product, and there-fore, they weren't needed in the first place. "Only about 20 percent of the features and functions in typical custom software are used regularly. Something like two-thirds of the features and functions in typical custom software are rarely used." [4]

Extra features are mainly owed to the "tactic" of traditional processes for early requirement specification. As described above, in traditional projects the requirements are being specified early in the development process. However, at the beginning of a project, the customers usually don't know what they really want, so a common fact is to request more things than they actually need. As a result, once these (unnecessary) things get formed into a requirement, they become an instance of overwork – especially if they also get integrated into the system's design. Furthermore, due to the long lead-times of traditional projects and their absence of customer collaboration and feedback, it is very common these unnecessary requirements to end up being implemented into the final product.

However, there is a huge cost in developing extra features. Apart from the obvious reasons of extra time and cost for implementing them, they also add complexity to the code base that drives up its cost at an alarming rate, making it more and more expensive to maintain, which eventually results in dramatically reducing its useful life. "Every bit of code that is there and not needed creates complexity that will plague the code base for the rest of its life." [4] Moreover, unused code still requires unnecessary testing, documentation and support. It does its share of making the code base "brittle" and difficult to understand and change as time goes on. Hence, extra features not only have a negative impact on the development cost of a software product (by means of additional time and resources for implementing them), but also on its overall lifecycle in general (in terms of reduced adaptability and increased maintenance costs). That's why the cost connected to them dominates all other costs in software development.

Lean's theory encounters this major productivity and financial bottleneck by providing better control of over-engineering. That is mainly achieved with the principle *Decide as Late as Possible*. Complementing to the definition given above, decide as late as possible argues that organizations should decide only upon immediate, clear issues, while leaving secondary and/or uncertain decisions for later.

In Lean's context, if developers code more features than are immediately needed, is considered as waste. The best opportunity to eliminate such waste is by writing less code. According to Poppendieck [4], in order to write less code, the developers need to find the 20 percent of the code that will provide 80 percent of the value and write that first. As given above, by value is meant customer and product value. Hence, what Poppendieck [4] argues for is that developers should first identify and implement the features which are most demanded by the customers, since they are the ones that provide the most value both to the customer and the product. In addition to that, since the most-valued features usually constitute the "core" of the system, it is unlikely that they will change during the project, while less-valued, secondary features are more probable to change or become obsolete in the project's duration. Once all of the high-demanded features get implemented, the developers shall continue adding more features, stopping when the value of the next feature set is less than its cost.

An additional benefit of implementing the most high-demanded (and thus, high-valued) requirements, is that it helps in maximizing the flow of delivered value ("*Add nothing but value*"). The idea that flow should be "pulled" from demand is fundamental to Lean development. "Pull" means that nothing is done unless and until a downstream process requires it. The effect of "pull" is that development is not based on forecast; commitment is delayed until demand is present to indicate what the customer really wants [20].

In a nutshell, decide as late as possible argues that organizations should only commit to the activities which are highly demanded by the customer, since they provide the most value to the product, while delay deciding upon activities that include uncertainty, as they are more likely to change – it is always easier to make a change of something that is not yet specified in detail than of something that already has been implemented. The combination of implementing the most highly-valued requirements and developing in small increments (increased customer responsiveness and collaboration) ensures that only the most important and necessary features get implemented into a system – or at least minimizes the risk of implementing extra features. By that is meant that even if extra features get implemented, they won't be so many (in number) or that unnecessary; they will still provide some value to the product, even if it's not the "optimal". This is mainly owed to continuous assurance (which results from increased customer collaboration), because it enables the customers to also verify if everything which have been implemented is what they expected. Moreover, by involving the customers in the development process, makes it easier for them to realize if what they initially asked for, still applies (since they are actively participating in the project) and in case it doesn't (i.e. if they have changed their minds about a feature), the organization will be notified on time. In summary, Lean software development provides better control of over-engineering, which – according to Lean's theory – assures closer fit to the real customer needs and reduces the cost of a software product (by minimizing the risk of implementing extra features).

Both Ericsson and Tieto described extra features as one of the main drawbacks of their previous traditional processes. In particular, some of the most negative impacts identified were high development and maintenance costs and low productivity. This, in fact, was one of the main reasons why the two organizations decided to change from traditional development to Lean.

According to Lean, the key to avoid over-engineering (extra features) is development to be based on customer demand; organizations should focus on implementing only the most important and "pressing" requirements, while leaving the secondary ones for later. This was adopted by both Ericsson and Tieto as part of their Lean implementation process. More specifically, when Ericsson embodied Lean, the concept of demand-based development (which derives from the principle *Decide as Late as Possible*) was converted into continuous requirement prioritization.

As described above (see Section 4.1), in SD all the requirements of a product are gathered in a repository, where they are prioritized based on their value and importance (for the customer and the product). When a suitable number of highly prioritized requirements that can be combined into a requirements package is available (based on that they fit well together, etc.), a new project is initiated. The project's length is a limitation to the size of the requirements package – the requirements from the package should be able to get implemented within the project boundaries (i.e. in about 3 months). Once the first set of requirements get implemented, the remaining ones in the repository get re-prioritized and re-packaged so the next project can start. Since the inflow of new requirements is constant in Ericsson (owing to the continuous evolving telecommunications domain), it is important for the organization to continuously prioritize the requirements of the repository, in order to always choose the requirement packages most suitable for implementation (considering dependencies, cost, market window, etc.). Additionally, since in Ericsson's projects the requirements are prioritized towards the current baseline (system version), their priority is updated every time the baseline changes (i.e. when a new system version is released). As described by Tomaszewski [8], continuously prioritizing the requirements is important because changing the baseline may actually change the importance of a requirement.

The lead benefit for Ericsson from continuous prioritization of their requirements is the assurance that only the most "pressing" requirements are being implemented for each new release of the system, and hence, each new release is highly demanded by their customers (increased customer satisfaction and acceptance). Additionally, by implementing only the most pressing requirements, increases the likelihood that requirements with lower priority which are more likely to become obsolete (unnecessary, outdated) are not yet implemented when the customer or market demands change. Therefore, the work and cost for implementing them will not become wasted.

The technique of continuous requirement prioritization is also being used by Tieto, with similar results/benefits. Moreover, another measure of Tieto against extra features is customer verification. In D2M projects (see Section 4.2 for a description of D2M) at the end of each iteration the customers verify if the implemented functionality is what they asked for. This enables the organization to identify if any unnecessary features have been implemented, or if any requirements have been accidently misprioritized. "If we have done something which is not important from customers' point of view then it's an overwork." (Project manager – Tieto) In Lean's context, overwork is considered as waste – you worked more than you should. As stated during the interview with Tieto, in their previous process, overwork (in terms of extra features) was usually owed to misunderstandings between them and the customer. When the company adopted a Lean thinking, the length of their iterations got reduced (as indicated by the principle *Deliver Fast*), which resulted in more frequent customer feedback. Part of that feedback is assurances (confirmation) that the implemented functionality coincides with what the customers asked for. This is called customer verification. Customer verification established better communication channels between Tieto and their customers, which effectively reduced the amount of such costly "misunderstandings".

In conclusion, the implementation of the "Demand-based development" concept (which originates from the principle *Decide as Late as Possible*) as continuous requirements prioritization, helped both organizations of this study to avoid extra features in their projects, by focusing the implementation of a project's requirements based on their importance and value. Moreover, customer verification (which is an outcome of the *Deliver Fast* principle) provided an additional layer of protection against extra features, by identifying if any unnecessary requirements have been accidently implemented. Hence, the adoption of Lean's principles by both Ericsson and Tieto resulted in effectively minimizing the risk of extra features in their projects, which in turn, reduced the cost of their software products (both in terms of development and maintenance costs).

5.2. Quality-improvement

Quality is also a very significant component in software organizations. This subsection draws the quality improvement picture of Lean in two phases: from a theoretical view and a practical view. The combination of those two aspects eventually results in validating Lean's theory from a quality perspective.

Quality cannot be added by focusing on just one principle; five principles of Lean software development together increase the quality of software:

- Eliminate Waste
- Empowering the Team
- Build Quality In
- Decide as Late as Possible
- Optimize the Whole

Eliminate Waste

Mensuration of cost will not enhance the quality since the quality cannot be heightened by evaluating the cost. To cognize what is wrathful, first it's necessary to know about what is destructive since nobody visualizes their work as a waste. Identifying waste by capable planning; communicating, testing and maintaining the process lifecycle, normally adds value to the product. Planning should involve all considerable facts and name all the waste before the project start. Taking the extra cost by eliminating waste, normally minimizes planning by assuring to do it correct in the first time. When a proper planning is involved in a project, then the quality of the product increases .When all the waste is removed from the process and if waste is known by the development team, then the development gets faster and the quality of the end product increases automatically.

In both Ericsson and Tieto had problems with their traditional processes; for example a considerable amount of time was spent for making "routine" paperwork. In many cases those documents had no use. While maintaining this documentation standard, the main quality of the product gets down for less implementing time. This was a common problem both organizations faced while using traditional development processes. Lean assisted both organizations in minimizing their documentation.

Empowering the Team

A drivable principle of Lean software development is to take decisions down to the people who actually append value to the product. Lean is not a rigid or stiff process; it's a process with planning and respect. Lean software development gives priority to people and collaborating team work. It focuses on forming and encouraging teams to address and resolve their own problems. Lean software development influences the workers to use their tacit knowledge in team work. The result of using tactical management increases communication and workflow. Waste is a big issue for quality; communication with the people who already know about the project is far easier than with the person that doesn't have knowledge about it. It's very important for the people to know their roles and understand them, so they need the knowledge. With *Empowering the Team*, Lean simply makes it easier for the managers.

Empowering the team helped both organizations in following up the quality steps. While the team has knowledge about the project and process, then it automatically adds value with several activities. After introducing Lean, both organizations gained a high level vision with understanding of the overall goals and advantages. Both Tieto and Ericsson gained team structure and the decision was to have cross-functional teams to share knowledge. This organized teams introduced many advantages. For example, in Ericsson stable teams allow an efficient project start and people enjoy improving their working process comparing to their previous team experience.

Build Quality In

Build quality in starts with the planning of the project, so it actually starts before the project. The goal of this principle is to build quality inside the code by eliminating waste and empowering the team. There are many ways to define the defects so the quality gets build in. If the product really needs quality, then it's important to inspect before the defect occurs. In order to do that, planning and discipline are needed. Nowadays there are too many tools to do that in a cost-effective way. Damn (2007) proposes the use of a measurement that takes into account that there is a particular phase where it is more effective to find a defect. The idea is to have a queue with no defects so it fulfills the customer requirements. In Lean it is very easy to follow the "plan-do-check and act" process. In this way there is no waste and when it's done with the check, it has already built the quality inside. Additionally, with iterations on each phase, each principle becomes more effective and clear with the continuous improvement process. However, this is only possible if the environment or the development organization has good communication and discipline and if Lean works in a disciplined way.

Ericsson was having problems when a product was developed aiming for a broad market launch. It was common that some customers asked for adaptations of the product which were not included in the main release. In the company's last working process, an adaptation project was initialized when the requested functionality had been implemented in a branch of the system and the product was released to the customer without integrating the adaptation part. That was delivered separately. It is clear that those projects faced problems with integrity and this is a crucial attribute for the quality of the delivered product. When Ericsson introduced Lean as their development process, the above problem was solved by involving smaller increments and by building quality in principle. Lean works with the product in total and tries to avoid sub-optimization. It doesn't see the product part by part but instead, as a complete product during implementation, testing and integration.

Decide as Late as Possible

This principle of Lean focuses on three dimensions: meeting the customer requirements, pulling from demand and maximizing the flow. The principle also covers amplifying learning. In software development, the risk of exaggerating the details that customers require and prioritize them depending on their demands is not easy.

However, Lean’s approach came from the idea of "Do it right the first time". The process has the provision for customer to make changes and acceptance testing with customer requirements. So Lean follows the process of pulling and prioritizing the requirements. When this is done, the work flow is maximized because requirement testing is done before coding. Therefore, deciding late ensures maximization of the work flow and requirements to be pulled by the customer, which in turn, results in faster and hassle-free delivery with higher customer satisfaction. In this way, it also follows just-in-time delivery of products which provide additional value to the organization. The main and most effective ideas of Lean development is Just-in-time delivery and pulling requirements from demand, where its main benefit is presenting increments of real business value in short time period.

In the world of technology, the modern market environment faces difficulties when delivering a product. During this study, it was made clear that both organizations had problems with delivering products while working with traditional processes. Traditional development cannot help with faster delivery. The adaptations of Lean helped both companies to overcome this by deciding as late as possible, delivering faster with short-term increments and having morning meeting for evaluating their strategy. In order to accomplish faster work flow, sub-optimization needs to be avoided which results in just-in-time output with customer satisfaction. It is a lot easier to measure the cost of a software than its quality. Lord Kelvin (1889) mentioned: “When you can measure what you are talking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.” In software development, the quality of a software cannot be measured in numbers. Quality of software depends on customer satisfaction (how well the software is designed and how well it fulfills the customer requirements).

Optimize the Whole

Optimize the whole means that there is no sub-optimization of the work. Lean software development is mainly driven by time, and sub-optimization is a weakness for any development process. Deeming the scope and purpose of a project from its beginning to its end is very important. Maintaining measurement with parts by parts should be avoided. In software development it is said that nothing is complete until it’s fully complete. Lean is usually structured around teams that hold responsibility for the overall project and this adds value to the end product. From the beginning to the end, everybody related to the project knows their role and responsibilities. So by optimizing the whole, Lean adds value and quality to the product and delivers a complete product as a result.

Both organizations got improved after adopting Lean’s principles. Lean helped them in identifying many quality attributes which added value to their end product(s). In the case of Ericsson, while they were working with their old process, they had problems with flexibility, usability and efficiency. When an organization deals with this sort of problems, then it follows up with maintainability and usability issues of the delivered product which is a threat to its quality. Lower quality products with higher development costs result in unhappy customers. In the case of Ericsson, Lean helped the organization to increase interoperability on its processes and communication with their customers.

As described in the interview, in the case of Tieto, if they worked in four-month long iterations, then after each iteration they would have to spent several weeks in order to fix the problems that occurred during the iteration, which is considered as overwork. As described above, overwork is a form of waste which keeps an organization from adding quality to the end product. However, Tieto is now working having continuous integration in mind, which helped them into solving lots of problems and producing products of higher quality compared to their last implemented ones. Focusing on quality attribute not only adds value to the end product, but it also makes the architecture of the software easy to understand and minimizes the cost for maintenance. Lean development helps to develop easy-to-use systems which are easier to understand and hence, they don’t need huge amount of documentation. As a consulting company, Tieto always wanted to add extra value to their customers and by adopting Lean, helped them in fulfilling their requirements.

6. Conclusions

The goal of this study was to assess the premise of Lean Software Development. Namely, Lean is a process that promises to reduce the development cost, all the while increasing the quality of the end product, through seven principles. The assessment was carried out in two stages with the first one being the theoretical part where after an extensive bibliography research, a hypothetical conclusion - as to the validity of the premise - was reached. The aim of this stage was to see how Lean Software Development can (in theory) reduce the cost of a software product and increase its quality. Through this process it was elicited that all the seven principles (collectively) play a role in achieving those goals. In other words, no one principle can stand by itself for either reducing the cost or increasing the quality of a software, instead all seven of them “complete” each other for achieving that.

The second stage was the empirical stage where representatives from two major companies that have implemented Lean (or a process inspired by Lean) were interviewed which along with a case study review gave an insight on the practical uses of Lean and the adaptations thereof. The findings provided for two sets of data; the cost-efficiency of Lean, which was analyzed in numbers, and the quality-improvement of it, that cannot be measured as it is something subjective. The aim of this stage was to investigate whether Lean can - in practice - reduce the cost and improve the quality of a software product. The results of the investigation are the following:

Cost-related results
• *Waste elimination* (know what waste is and plan to avoid it).
• *Shorter projects* (less exposed to the risk of changing requirements).
• *Requirement prioritization* (based on importance and provided value).
• *Customer collaboration* (rapid feedback on the correctness of estimations – avoid rework).
• *Delayed commitment* (better control of over-engineering)

Table 6.1 – Cost-related results

As described above (see Section 2.2) Lean looks at the development process from a value perspective. Due to that, organizations working according to Lean are able to identify activities which are keeping the value from being delivered to the customer and adding unnecessary costs to the product (e.g. excessive documentation, late testing, etc.). Confirmed from the interviews (see Section 4), with that knowledge an organization is able to better plan and design its projects in order to exclude such activities, resulting in lower development costs and higher delivered value. As shown in Section 2.3, Lean projects consist of short development cycles. Small projects tend to have better controllability since they enable more correct predictions and estimations regarding their lead-time and cost. In short projects, progress can also be monitored more easily. Additionally, short projects are less exposed to the risk of changing requirements, since needs are less likely to change in a short time period. However, short cycles are not the only measure of Lean against changing requirements. Lean stresses the importance of including the customer early in the development process. As in Ericsson's case (see Section 5.1), early customer feedback enables continuous requirement prioritization based on their importance and provided value, ensuring that only the most "pressing" and valued requirements are being implemented in each project increment (demand-based development), postponing implementation of secondary, less-valued requirements which are likely to change within time. Additionally, early customer collaboration provides for rapid feedback on the correctness of estimations, which makes possible to detect "misunderstandings" between an organization and the customer much earlier, reducing the risk of implementing inadequate functionality. Reinforcing Lean's ability against over-engineering comes the principle Decide as late as possible. As described above (see Section 2.3) Lean believes that an organization should only commit to the activities which are highly demanded by the customer, since they provide the most value to the product, while delay deciding upon activities that include uncertainty, as they are more likely to change within time. Additionally, if a decision must be made over an issue which is unclear, there should be a "back door" for reversing it later on, if needed. In all organizations under study (see Section 5.1), delaying commitment and making reversible decisions enabled to build a capacity for change into their systems by maintaining options, effectively reducing the risk of implementing extra features which, in turn, would have added unnecessary costs and complexity to their products.

Quality-related results
• *Waste elimination* (have waste in mind when planning and designing a project).
• *Build quality from start* (take quality under consideration from the initial phases of a project, so quality gets "built-in" the architecture).
• *Frequent releases* (meet customer needs faster and deliver value to the customers earlier).
• *Late decision-making* (work flow maximization and high customer satisfaction).
• *Cross-organizational awareness* (shared knowledge and experiences).

Table 6.2 – Quality-related results

As described above (see Section 2.3) the "cornerstone" of Lean thinking is waste elimination. Confirmed from the interviews (see Section 4), all organizations working according to Lean are trying to identify and eliminate waste before it occurs, from the initial phases of their projects. By having waste under consideration when planning and designing a project, the waste is removed (or at least minimized) from the process resulting in faster development and higher end-product quality. However, waste is not the only thing that's been taken under consideration early in Lean projects. Lean strives that quality also needs to get integrated into the system from start. As shown above (see Section 5.2) all the organizations under study were having quality in mind from the initial phases of their projects, so quality to get "built-in" the architecture, which results in enhanced product quality. Identified from the interviews (see Section 4), one of the major changes that the organizations under study experienced when switched into Lean, was shifting from the enormous, long-term projects of traditional development into the short increments of Lean development. Lean projects consist of small iterations where, at the end of each iteration, a new version of the product is released. Frequent releases enable early customer feedback which, in turn, ensures that the end-product gets closer to customer expectations, resulting in meeting customer needs faster and delivering value to the customers earlier. Despite the ”urge” of Lean for rapid development and early releasing, when it comes to making critical decisions, Lean prompts for postponing them for as late as possible. Deciding late ensures maximization of the work flow and requirements to be pulled by the customer, which in turn, results in faster and hassle-free delivery with higher customer satisfaction. Lastly, a principle of Lean indirectly related to quality is Empowering the team. Lean stresses the fact that decisions should be taken down to the people who actually append value to the product. In order for that to work out well, an organization should have a high level vision with every member involved within a project being aware of the overall goals and advantages. When the team has knowledge about the project and the liberty to address and resolve their own problems (make decisions), it automatically adds value with several activities. In both organizations under study, cross-organizational awareness helped the companies to build more stable teams by encouraging the workers to use their tacit knowledge in team work (see Section 5.2). Stable teams allow an efficient project start and people enjoy improving their working process by sharing knowledge and experiences.

The overall conclusion from this study was that Lean Software Development is a cost-efficient and quality-improving one. Lean, through its various principles, manages to cut back on the workload, thus promising less costs, and to include the end product user (in this case the ordering company) in the development process, thus promising quality enhancement.

An additional conclusion that can be drawn from this study is that Lean is not for everyone. That is to say that as the two companies from this study (Tieto and Ericsson) had tailored Lean to their needs, it is perhaps wiser for most companies to follow that example as no rigid development process would ever prove to be successful (since no two companies have the same needs) and it would have to be bent to each company’s needs to be profitable.

The results of the investigation presented in this paper provide for an aid for any company who is on the fence about adopting Lean and are unsure about its theoretical or practical uses.

7. References

[1] *Selecting a Development Approach*, 2005, http://www.cms.hhs.gov/

[2] Taiichi O, Norman B, 1998, *Toyota Production System: Beyond Large-Scale Production*

[3] Poppendieck M, Poppendieck T, 2003, *Lean Software Development: An Agile Toolkit for Software Development Managers*

[4] Poppendieck M, Poppendieck T, 2007, *Implementing Lean Software Development: From Concept to Cash*

[5] Highsmith J. et al., 2001, “The Great Methodologies Debate: Part 1”, *Cutter IT Journal*, Vol. 14, No. 12

[6] Highsmith J. et al., 2002, “The Great Methodologies Debate: Part 2”, *Cutter IT Journal*, Vol. 15, No. 1

[7] Adomauskas V, Murauskaite A, 2008, *Bottlenecks in Agile Software Development Identified Using Theory of Constraints (TOC) Principles*

[8] Tomaszewski P. et al., 2007, “From Traditional to Streamline Development – Opportunities and Challenges”, *Software Process: Improvement and Practice*, Vol. 13, pp. 195-212

[9] Creswell J. W, 2003, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*

[10] *Lean Management Series*, 2009, http://www.unisa.edu.au/

[11] Taylor W. F, 1911, *The Principles of Scientific Management*

[12] Womack J. P. et al., 1990, *The Machine that Changed the World: The Story of Lean Production*

[13] Jenner R. A, 1998, “Dissipative Enterprises, Chaos, and the Principles of Lean Organizations”, *Omega International Journal of Management of Science*, Vol. 26, No. 3, pp. 397-407

[14] Liker J. K, 2005, *The Toyota Way*

[15] *Quality with a Name*, 2006, http://www.jamesshore.com/

[16] Kilpatrick J, 2003, *Lean Development – A team approach to Software Application Development*

[17] *Lean Software Development*, 2008, http://railsstudio.com/

[18] *Estimating the learning curve*, 2008, http://agilecommons.org/

[19] Mehta M. et al, 2008, “Providing value to customers in software development through Lean principles”, *Software Process: Improvement and Practice*, Vol. 13, No. 1, pp. 101-109

[20] *Principles of Lean Thinking*, 2002, http://www.poppendieck.com/

[21] *Seven Principles of Lean Software Development – Deliver Fast*, 2008, http://agilesoftwaredevelopment .com/

[22] *Waste & Value*, http://www.objectwind.com

[23] Dalcher D, Brodie L, 2007, *Successful IT Projects*

[24] Yin R. K, 2004, *Case Study Research: Design and Methods*

[25] Neuendorf K. A, 2002, *The content analysis guide book*

[26] MacCormack A. et al, 2003, "Trade-offs between productivity and quality in selecting software development practices", *IEEE Software*, Vol. 20, No. 5, pp. 78-85

[27] Johnson G. et al, 2005, *Exploring Corporate Strategy*

[28] Sommerville I, 2004, *Software Engineering*

[29] *Software project failure costs billion. Better estimation & planning can help*, 2012, http://www. galorath.com