Using Lean Principles to Improve Software Development Practices in a Large-Scale Software Intensive Company

University of Oulu
Department of Information Processing Science
Master’s Thesis
Riku Suomela
22.10.2015
Abstract

Lean software development is the result of adapting lean principles from the manufacturing context to the software development domain. Recently, the various applications of lean software development have been studied but more empirical evidence is needed, especially from the practitioners’ point of view.

Firstly, this thesis provides answers for the understanding of lean software development from the practitioners’ point of view. Secondly, this thesis provides answers on the opportunities and barriers in applying the lean software development. In order to study this, a case study was conducted in a large-scale software intensive company. Focus groups were conducted to collect qualitative data.

Studying the understanding of lean software development showed that four of the seven lean software development principles were identifiable from the discussion in the focus group sessions. The difference between agile and lean was recognized.

The opportunities in achieving a culture of continuous improvement and involving people in the transformation were found and can be also identified from the existing research. Some new opportunities were also identified, such as using informal code-reviews as a practice in development and focusing improvements on the activities that consume the most time in the day-to-day work.

The barriers that were found, such as avoiding sub-optimization, facilitation of improvement and having time to experiment, can also be identified from the existing research. Some of the barriers not identifiable from the existing research were the lack of quality thinking and varying standards in gate keeping. The findings of this study were presented in the case company with positive feedback and were discussed to be included into future improvement initiatives. This study also identified the power of the focus group method as a tool that could be used to drive improvement work.

Suggested directions for future research include studying lean software development in a similar case study and taking a look at the possibilities of using focus group method as a tool for driving improvement initiatives in software development companies.

Keywords
Software, Lean, Agile, Lean Software Development, Product Development

Supervisor
Professor Jouni Similä, Teemu Karvonen
Foreword

Firstly I want to thank my supervisor Professor Jouni Similä for all the shared wisdom and guidance during the process of making this thesis. Secondly, I want to thank Dr. Pasi Kuvaja and Teemu Karvonen from M-Group at University of Oulu for their continuing support.

I, of course, owe a special thank you to my family and my girlfriend for the love and trust they continue to have in me. Without you, I probably would’ve lost my mind!

The process of making this thesis has taught me what it means to do academic research. Learning to plan, conduct and review research are lessons that I have learned and appreciate enormously.

Riku Suomela

Oulu, October 2015

“Sometimes you just do things!”

- Scott Jurek, Eat and Run: My Unlikely Journey to Ultramarathon Greatness
1. Introduction

Lean software development is a separate methodology from agile software development. However not much is known about what it actually is, how it could be applied and how it is understood by practitioners. The purpose of this study was to explore lean software development through a case study in a large-scale software intensive company. However it should be noted that lean software development cannot be studied in isolation and it is needed to reflect on a variety of lean literature.

A large-scale software intensive company was interested to know more about possibilities in applying lean software development. Recent research has also recognized the need for further research on this topic (Rodríguez, Mikkonen, Kuvaja, Oivo & Garbajosa, 2013). The personal motivation for this research rose from the fact that the author had done his bachelors thesis on the topic of “Agile and lean self-assessment methods for software development companies”.

The term lean was made famous by Womack, Jones and Roos (2008) in their classic book *The Machine That Changed the World*, where the Toyota Production System was studied and explained. The ideas presented in this book were then analyzed and summarized in *Lean Thinking: Banish Waste and Create Wealth In Your Corporation* by Womack and Jones (2013). Later on a distinction between Lean Manufacturing and Lean Product Development has been made. Lean Software Development is then again separated as its own research area from Lean Manufacturing and Lean Product Development. Poppendieck and Poppendieck (2003) present the foundational ideas for Lean Software Development in their book *Lean Software Development: an Agile Toolkit*. They have also published two more books on the topic, taking a bit different approaches. These books are called *Implementing Lean Software Development: From Concept to Cash* and *Leading Lean Software Development: Results Are Not the Point* (Poppendieck & Poppendieck, 2006; Poppendieck & Poppendieck, 2009). Rodriguez et al. (2013) have studied the strengths and challenges of building lean thinking in software development organizations. Rodriguez et al. (2014) have also taken a look at combining lean thinking and agile methods in software development. A systematic mapping study has been conducted to study lean approaches to large-scale software systems development by Pernstål, Feldt, and Gorschek (2013).

The research questions for this study rise from the need to understand possibilities in applying lean software development. To answer this need, three distinct research questions were formulated. The first research question was: “What is the practitioners’ understanding of lean software development?” The second research question was: “What are the opportunities for applying lean software development?” Finally, the third research question was: “What are the barriers for applying lean software development?” With the first research questions we assess the understanding of practitioners on lean software development to be aware of various different interpretations on the topic. This is important because if, for example, interpretation varies a lot then it clearly indicates that more effort needs to be put on sharing the knowledge on the topic and ensuring that people mean the same things when discussing about lean software development. This is important for the academia, also. The other two research questions are counterparts. First of all we want to know the best opportunities and possibilities for applying lean software development but at the same
time we need to be aware of the biggest barriers that create challenges in the application process.

To tackle the research questions mentioned in the previous chapter, the case study research method was selected as the most appropriate one. The selection of case study was based on the need for knowledge about the real situation with lean software development. Runeson and Höst (2008) provide guidelines for conducting and reporting case study research in software engineering. A large-scale software intensive company was selected as the case for this research. For the data collection, focus group method was selected. Kontio et al. (2014) have studied the practitioner and user experiences of using the focus group method in software engineering research. The reason for selecting the focus group method was that with this method new idea generation is possible when a group of people gathers to discuss a focused topic over a few hours of time. Because the topic of lean software development is largely based on driving lean thinking, this kind of idea generation is needed to get knowledge on the topic. When compared to, for example, a group interview, the benefit of organizing a focus group is that it allows the participants to discuss the topics more freely; it creates a relaxed atmosphere and allows for free flow of thinking between the participants. This is especially important on a topic where we are not expecting any specific, direct answers. When practitioners feel that they are given the control, they actually contribute more.

The contribution of this thesis can be divided into two. Firstly it contributes to the body of knowledge on lean software development. In this sense this study confirms and challenges existing research on the topic and also points out new directions for future research. Secondly it contributes to the practical need in industry to know how to apply lean software development.

This thesis is structured as follows. First a literature review on lean, lean software development and related topics is presented to introduce the reader to the background of the topic. This background contains lean manufacturing, lean product development, lean transformation, lean software development and the relationship of lean and agile software development. After this the research methods will be presented to describe what methods were used, how the research was designed and eventually conducted. Then the findings from the focus groups are presented by framing them with the three research questions, respectively. After the findings have been presented, they are discussed further by reflecting with the relevant literature. The thesis ends with conclusions, research limitations and suggestions for further research. The structure of this thesis is linear-analytic, which is the standard research report structure (Yin, 2009).
2. Lean

This chapter explains lean because it is the fundamental concept being handled in this thesis. Because of this importance, it is necessary to present the current research on lean and related topics thoroughly. First the origin of lean and where it originated from, the manufacturing environment, is explained. Then the fundamental principles that lean represents are explained. Because traditionally lean is thought of as something only related to manufacturing, a distinction is made between lean manufacturing and lean product development by presenting the latter in more detail since it holds relevance to the topics of the thesis. Previous research on lean software development is opened up with the related principles explained. Next a distinction is made between agile software development and lean software development because they have some overlapping which needs to be explained. After this, Enterprise Transformation is explained to help understand what the difference is in discussing lean in software development and on enterprise level. Finally systems thinking is also presented in its own sub-chapter because it can be used as a tool when moving forward with lean software development. Figure 1, The House of Lean, sums up the main characteristics of lean as well as lists the 14 principles of lean by Larman and Vodde (2009). It should be highlighted that the reason for introducing so many different aspects of lean in this chapter is because fuzziness still lies on top of the whole concept of lean. Because of this the relations of lean manufacturing, product development, systems thinking, agile software development, enterprise transformation and lean software development are essential to be mapped and explained to understand how these pieces fit in to the bigger puzzle.

A systematic literature review on Lean Software Development has been conducted recently (Jonsson, Larsson, & Punnekkat, 2013). In that review, literature was searched systematically for seminal sources on Lean, Lean Product Development and Lean Software Development. Nine seminal sources were identified each presenting slightly different principles but the basic thinking behind all of these was found to be the same and aiming for the same results. The seminal sources were divided to three groups: Lean Production, Lean Software Development and Lean Product Development. In Lean Production group the sources were three: Womack, Jones and Roos (2008), Liker (2004) and Ōno (1988). For Lean Software Development five sources were identified: Poppendieck and Poppendieck (2003, 2006, 2009), Anderson (2010), Middleton and Sutton (2005), Coplien and Bjørnvig (2011) and Morgan (1998). And the last group of the three, Lean Product Development, consisted of one source: J. Morgan and Liker (2006). In this thesis, lean production is explained by using Womack, Jones and Roos as the main source because it is widely seen as the classic and covers all the relevant aspects of lean production. For lean product development it was decided to use the theory by Ward and Sobek II (2014) over the one by J. Morgan and Liker (2006) because even though the frames they present their theories in are different, the content is more or less the same and gives a good overview of lean product development. For lean software development it was decided to use the theory by Poppendieck and Poppendieck (2003, 2006, 2009) because it is widely recognized and has the most references in Google Scholar over the other sources mentioned earlier. It was also identified in the aforementioned literature review that Poppendieck and Poppendieck (2003, 2006, 2009) had the largest number of references from primary papers over other sources on lean software development (Jonsson et al., 2013.)
2.1 History

The term lean is coined from the book called *The Machine That Changed The World* by Womack, Jones and Roos (2008). This does not mean that the idea of lean thinking was born at that time but it was actually born way before in the Japanese automotive industry, especially Toyota.

The story starts from Henry Ford and the western automotive industry. Henry Ford was the person who introduced the systems he called mass production. It was a counter-action for the inefficient and slow manufacturing that was happening in craft production where each product was “hand-made” and individually crafted. Ford introduced techniques that would revolutionize manufacturing, as it was previously known. Ford Model T was the first car that was designed for manufacturing and at the same time it had value that the customers were willing to pay for. For mass production to be successful, the core idea is not only to have a continuously moving assembly line even though it is usually important. The biggest innovation is actually the fact that all the parts that are required to build a product are interchangeable and can be attached to each other with ease, no crafting needed! This is the fundamental change that was needed in order for an assembly line to really work continuously and effectively. Henry Ford also introduced the interchangeable work force to the assembly line by organizing the work in a way that each assembler on the line had one simple job, for example putting a couple of bolts in, and nothing else. This guaranteed that a new assembler could be taken in and trained in no time and each worker was replaceable. All the tools and machines in the plant were also designed to do one job and do it efficiently, cutting expenses of parts. The core problem that Henry Ford faced was that he was not able to introduce much variability to his products. Even though the Ford Model T had the possibility of having some variability such as couple of different body styles, it was little. The chassis stayed identical for almost 20 years, through the whole production of Model T. This meant that the specification was the same and didn’t change. The success of Ford cars was based on low prices which, they were able to deliver because of the mass production way of manufacturing over the previous craft production way. (Womack et al., 2008.)

Lean production is the advancement from mass production. It literally is what the difference in the naming tells you; production done leaner so that when you cut the extra mass out you get lean. Just as if you lose weight and get rid of that extra load that’s being carried, you become lighter, more efficient. Eiji Toyoda and Taiichi Ohno are the two names of Toyota that can be credited for introducing lean production. In 1950 Eiji Toyoda visited Ford manufacturing plant to study the mass production system there. After he returned from this trip he and Taiichi Ohno studied the possibilities of applying the same mass production system for Toyota, in Japan. However after initial studies on the applicability of such a system, they realized that it would never work in Japan as it did in the West because of Japan’s isolation and certain characteristics in the culture. The system that they eventually created became known as the Toyota Production System, lean production. (Womack et al., 2008.)

So how was lean production arranged in practice? Toyota focused on making the production flow through the whole value stream of the product. They introduced machines that could be easily moved around the plant and re-organized and the same machines could be used to do not only one but several different tasks. Andon cord was introduced and workers were able to stop the production line at any time if a problem occurred. Andon basically means a system that allows workers to notify everyone else of on-going quality problems through different means. Pulling an Andon cord, for
example, triggers a signal such as a light turning red on top of the work cell. Kanban boards were used so that work could be done and delivered just in time and no massive stock of parts was needed at workstations. The team collectively embarked on efforts to improve the process. These efforts later on came to be known as continuous improvement, Kaizen. When problems or defects were faced in the production line, root cause analysis was carried out and a technique called The Five Why’s was used which simply meant asking five why-questions to really dig deep and get to the bottom of things. When the root cause got fixed once and for all, it yielded massive benefits when the problems were dealt with right there on the spot instead of firefighting the problems later in the development. All these things resulted in a massive decrease of rework needed at the end of the production line. The company was able to introduce greater variability in their products for the customers and simply higher quality was embedded in the cars being produced. Concentrating on quality and perfection from the start to the end was fundamental. In addition to the actual production line, Toyota had perfected their whole supply chain, which extended from the suppliers all the way to the customer. (Womack et al., 2008.)

Product development and engineering is no doubt an extremely relevant component in the process of creating high quality products. In mass production, product development was done by a team consisting of a weak leader (a person who just coordinated the team with no overview of the product being developed) and product engineers from multiple areas of specialty. These engineers knew their one thing and knew it well but when they had to work together to create a product that could be manufactured, problems occurred. Instead of organizing development work like this, Toyota had teams consisting of a leader with strong technical expertise of the whole product under development and engineers that were team players and worked well together. This meant that the teams were able to reap the benefits of working as a team. It is not teamwork if you work in the same office but never exchange ideas or discuss about them. (Womack et al., 2008.)

The differences between mass and lean producers in product development can be summed in four categories: Leadership, Teamwork, Communication, and Simultaneous Development. Leadership difference comes mainly in the form of Shusa, The Chief Engineer. Chief engineer at Toyota was simply a person who was responsible for developing a new product into manufacturing. This person was a seasoned expert with years of experience in leadership and expertise across wide range of functional areas. Teamwork difference comes in the form of tightly knit cross-functional teams working together to achieve a common goal by using their wide areas of expertise as a tool. During the development program these engineers stay under the control of the chief engineer. The difference in communication comes from the fact that in mass production product development the team commits to doing some plans without proper knowledge on the feasibility but in lean product development the team agrees as a group what they will commit to do instead of individuals. Also in lean product development projects, the number of people during the project does not increase but rather decreases because workers, such as market assessors, are no longer needed and can be dropped off to make the project leaner. The last difference comes down to simultaneous development. This means that the production of cross-dependent components can be started almost simultaneously because the people working on the design of these different components work together and know exactly what the other one is doing. Because of this, at Toyota for example, it was possible to start die production at the same time as the body design since the body design engineer was able to give his plans to the guy producing the dies needed to make the body. The result of this simultaneous work was obvious; dependencies between different production components didn’t necessarily mean that the
component having the dependency needed to wait for the other one to be 100% finished before it was possible to start the work on it. (Womack et al., 2008.)

Figure 1. The Lean thinking house. Redrawn based on Larman and Vodde (2009)

2.2 Lean Principles

After the book called The Machine That Changed the World (1990) was published James P. Womack and Daniel T. Jones aimed to deliver the core message of lean in a new book called Lean Thinking (1996). In this book they wrapped up the message of lean into five principles. These lean principles are presented in the following sub-chapters and are called Value, The Value Stream, Flow, Pull and Perfection.

2.2.1 Value

It is said that with lean it all starts with the focus on value and what it is. Value is something that is always defined only by the customer and hence all development work should be aimed towards creating increased value for the customer. A specific product or a service creates value when it meets customers’ needs at specific price at a specific time. This value is ultimately only created by the producer and this is what producers mainly struggle with: How to create the ultimate value for their customer. It sounds like an easy job to go and ask the customer what they want and then do it because that should create the ultimate value, right? With most companies it seems that this is not the case however and they rely on existing products and creating variants of these products which then serve the customers’ needs somewhat. Instead the companies should be asking the customer continuously what they want and then through joint efforts, create
the product or service that fits their needs and provides the ultimate value. (Womack & Jones, 2013.)

2.2.2 The Value Stream

Value stream means all the relevant activities needed to create and deliver a product. The value stream consists of bringing the product through three different management tasks, which are: *Problem-solving task* that runs all the way from concept work to launch, *information management task* that runs from taking in the orders and scheduling the delivery and *physical transformation task* which means transforming raw materials to the finished product. In lean, companies can identify this value stream for all of their products in order to be able to see all the activities taking place when creating a product. This activity is called value stream mapping and analysis and it means basically drawing a diagram consisting of all the activities taking place in that specific value stream. What lean then focuses on is the removal of waste, *muda*, from these value streams. Activities that are not necessary and do not add value to the product are considered as waste and should be eliminated. All activities required to design, make and order a product can be separated into three distinct categories. These categories are 1) activities that create value as perceived by the customer, 2) activities that do not create value for the customer but are required by the current system that is used for creating the product and so on cannot be eliminated right away (Type one waste) and 3) activities that do not create value for the customer and can be eliminated right away (Type two waste). When the category three activities have been eliminated as unnecessary, non-value adding activities companies are clear to proceed to the next step which is to tackle category two activities. This, however, requires much more consideration because they’re built-in to the system and used by people in their everyday work. The elimination of these activities can be eventually achieved through flow, pull, and perfection techniques all so relevant to lean thinking. The result of this is a more efficient value stream where more time is spent on creating value for the customer instead of creating waste. The value stream must be recognized over company boundaries. Sub-contracting and third parties create a lot of handovers and they are all part of the products’ value stream so when the value stream analysis is done, they should be included as well. *Lean Enterprise* is a term used to refer to this inclusion of sub-contractors and third parties, all relevant stakeholders in the product creating and delivery process. If waste is not tackled in this way then it is not systematic and it becomes sub-optimizing, which might mean that the desired benefits are not gained from this activity. (Womack & Jones, 2013.)

2.2.3 Flow

Flow is the third one of the original lean principles as presented by Womack and Jones (2013). After all wasteful activities have been removed from the value stream; it is time to re-arrange the remaining value-creating steps in the most optimal way. The goal is that everything flows smoothly through the value stream into one common direction. This then, for example, creates flow of materials in a production plant, flow of information between actors and flow of development work. In large companies, this often requires the re-organization of the whole company. Traditionally companies are organized into departments and functions and work gets done in batches. This is intuitive for people. What is counter-intuitive is the creation of flow for the whole value stream across the lean enterprise; flow that stretches out all the way to third parties and other collaborating actors doing their part in the value stream of a product. The way this is done is that instead of departments and functions, the focus should be to have a value-
creating process where departments and employees are arranged so that they can do their work in the best way possible; for themselves and for the company. When work flows across the lean enterprise, also the employees know and see what’s happening because their work does not end up in a queue but instead gets taken into use right away. This results as instant feedback on problems, for example. In practice, creating flow in the factory floor by positioning materials and tools in the most optimal way is different from creating flow in product development by keeping development teams in synchronization, but the core idea stays the same. Value-creating steps arranged so that minimal waiting takes place and people can get on with their work. The key here is to have the right things in the right place at the right time because if that is not the case and unwanted items arrive to places where, for example, they are not needed at that time, that can be considered waste.

![Figure 2. Principles of Lean](The Lean Enterprise Institute, Inc., 2015)

2.2.4 Pull

The fourth lean principle is called pull. It is the opposite of push effect. Push is what mass production creates but what does it mean in practice? When throughput times are long in production, order handling and product development, it means that the producers are basically incapable of reacting to customer demand in real time without extensive stocks. Because you are incapable of delivering products on-demand it means that you have to forecast demand and based on these forecasts mass produce products into stock. This is not an efficient way of working and when the customer demand suddenly changes as it usually does, vendors and producers end up with massive stocks of manufactured goods that they need to get rid of. Getting rid of extensive stocks happens in the form of sales and often the retailers are forced by contract to buy and sell old products so that the producers can get rid of the unnecessary stock. Lean promotes pull instead of push and in a nutshell it means that customers on all levels are able to pull products from the producers instead of the products being pushed to them. With long throughput times this is not possible of course but focusing on customer perceived value, elimination of waste from the value stream and making the lean enterprise flow happen, it is possible to achieve dramatic reductions in throughput times. This then
means that producers are able to react more efficiently to demand in real time. So instead of goods being pushed to customers they are able to pull them from the producers and doing this they do not need to wait weeks to get them. When customers know that they are able to get what they want on-demand, they become more predictable and the fluctuations created by sales go down. Also the producers gain business benefits from this when they can strip out of massive stock created according to forecasts. With mass production and push way of delivering goods, creating variability in the products is a real challenge because if you’d be producing all possible variants for the customer, your already extensive stocks would almost literally explode to every direction. When goods are being pulled and produced on-demand, the producer is able to offer a wide range of possibilities for the customer to customize their products and in this way create variants. Of course there has to be a limit to this too but compared to mass production the difference is the sort of night and day. It all seems quite natural that no good or service should be produced and pushed to some direction where no request for this kind of product has been made. (Womack & Jones, 2013.)

2.2.5 Perfection

Perfection is the fifth and last lean principle. This might seem strange but when you look back to the work taking place in the first four principles, the principle of perfection actually nicely ties them all together. Because in a lean enterprise great transparency exists and everyone can see everything, customers, designers, factory workers and subcontractors are all able to work together and make changes that benefit the enterprise. Identifying value, optimizing the value stream, creating flow and eventually having the pull effect all work together in a loop where the principles of perfection are the driving force for always getting better, continuously improving. Employees are personally interested in perfecting work processes because it almost always yields benefits that make their own work easier in addition to eliminated waste and increased efficiency. The continuous improvement efforts can take the form of Kaizen. Kaizen events can be used to create change. The Japanese term Kaizen means literally “change for the better”. Daily Kaizen means an event that can take place ad-hoc. This simply means that someone gets an improvement idea for the current process or way of working, and it can be proven that this change creates efficiency, change for the better. Another form of Kaizen is Kaizen Workshops in which a larger group of people working in the same area gather up for a day or two to find solutions for a specific problem or set of problems (Poppendieck & Poppendieck, 2006). The core idea that lies in the principle of perfection is that in order for a lean enterprise to truly blossom like it is meant to, every single person working across the value stream needs to be doing his/her part and seeking ways to improve the current system. (Womack & Jones, 2013.)

2.3 Lean Product Development

Lean product development can and should be explained separately from lean manufacturing but these both need to work well together, in order for the whole value stream of a product to work. The result is then a truly lean enterprise. Some of the key goals of lean product development according to Ward and Sobek II (2014) are: reduced development time and resource usage, reduced quality problems, increased innovation and reuse of production systems and parts. In a nutshell the goal is to learn fast how to develop high quality products. All of these are very familiar from the most basic principles already presented in this thesis. Compared to lean manufacturing, lean product development is little known and discussed about outside Toyota (Ward & II, 2014).
Dantar P. Oosterwal is the author of a 2010 published book; *The Lean Machine*. In this book he explains how Harley-Davidson embarked on a journey to revolutionize their product development process using lean thinking. The outcome of this lean transformation was the recognition of Harley-Davidson for their management excellence, capability to innovate and massive growth in the company’s profitability. So it can be said that Harley-Davidson is a prime example of a large-scale enterprise being successful in transforming to lean product development and proving that the transformation yielded massive benefits, unreachable with the previous way of running product development. (Oosterwal, 2010)

Oosterwal (2010) mentions also that systems thinking is an integral part of a learning organization. “The phrase ‘systems thinking’ implies thinking about the world outside ourselves, and doing so by means of the concept ‘system’“(Checkland, 1999). Systems thinking can be divided to roughly two categories: Hard systems thinking and Soft systems thinking. Hard problems are usually the sort that can be quantified whereas soft problems often relate to people and therefore are much harder to quantify. Oosterwal (2010) states that systems thinking is the cornerstone for organizational learning and that it integrates the other four aspects: Personal mastery, Mental models, Building shared vision and Collective team learning. Systems thinking is used especially for solving complex organizational problems. The problem is that in long feedback loops the benefits can only be recognized months or years afterwards (Oosterwal, 2010). System maps are then used to help with this kind of work. System maps are drawings and / or diagrams that map out the relevant system components and their relations. The core idea that emerges from driving systems thinking to improve the way organizations work is that one should not take the most obvious, short-term solution but instead problems should be studied in-depth and long-term effects should be measure and taken into account when making these decisions. The connections inside and between different systems holds the key.

Waste in product development is not the same as waste in manufacturing. Waste in manufacturing has something to do with physical things, such as over the top stocks. In product development most of the waste is knowledge waste, invisible. There are three types of knowledge waste: Scatter, hand-off and wishful thinking. Scatter waste means when knowledge in the product development is spread all over the place, scattered, for some reason. It means that people working in the development have to waste their time in the search for appropriate knowledge. Scatter has two related wastes, which are barriers to communication and poor tools. Barriers are blockers for the flow of information and poor tools are the result from too well defined processes that require these. (Ward & II, 2014)

The second main type of waste is hand-offs. Hand-offs are the result from the separation of responsibility, action, feedback and knowledge. What happens then is that people with insufficient knowledge end up making decisions because some knowledge, for example, disappears in hand-offs. One common example of hand-off waste is when people get assigned and removed from projects continuously. This means that they carry some of the knowledge out and it does not end up being handed off to the people staying in the project or starting in it. Useless information and waiting are two wastes associated with hand-off waste. Useless information waste can be the result of hand-offs because developers have to transfer their knowledge to the managers and managers to the executives. This means a lot of slideshows and spreadsheets to move knowledge only once. Waste of waiting occurs when people are required to done things in sequences. You have to complete activity A before you can start activity B and eventually proceed to C. This can result in teams waiting for others to complete their
work, just because things are said to be done in a determined sequence. Activities can and should be done in parallel, because it is possible and this removes the amount of waiting in product development. (Ward & Sobek II, 2014)

Wishful thinking is then the third main type of product development waste. This simply means that decisions are made without appropriate data and knowledge and in this manner wishful thinking is what happens in development. What happens in conventional product development process is that the development team picks the first concept and starts to develop it. When they face problems and challenges with the design of the solution, they just try to force it and the end result is a product of low quality and high expense. More time and effort should be spent to choose the baseline, the fundamental concept of the product. Waterfall development refers to the process of developing a product in isolated functions, sequentially. First the product is specified, then designed, then implemented and finally tested and launched. Testing to specifications is a related waste to wishful thinking and means that only specifications are used to determine testing of the product, not the reality and all possible test cases. This leads to problems not being found until the customer does. Discarded knowledge waste simply means that knowledge generated during the development that could be useful simply gets discarded and unused. (Ward & Sobek II, 2014)

To overcome the challenges of traditional development, set-based concurrent engineering (SBCE) can be introduced. In this way of developing the development team is continuously exploring for different options to implement their subsystems or manufacturing systems. The possible solutions are quickly and effectively analyzed so data is gained about the feasibility of these solutions. This is an opposite action to what happens in traditional development when a concept is quickly chosen with no data and usually stuck with until the launch. In SBCE the team starts further development of an idea when they feel enough data has been gathered and analyzed to justify the go-decision for development. All the data and results generated during the evaluation and analysis of solution alternatives are recorded and transformed into, for example limit curves so that further development can use these to avoid rework and assess the limits of new concepts. (Ward & II, 2014)

There are four underlying concepts for lean project management. These concepts are Cadence, which means that the development moves in a constant rhythm and this helps to organize various operations when there is this sort of predictability available. It can be seen as the pulse of development. The concept of Flow is probably one of the most recognizable in lean and it means that needed resources move across various functions and are available when needed. Pull then means the reaction to customer demand and delivering when needed but not more. Value-creating management refers to the fact that the management should also be directly creating value by being involved in the designing of the systems and working as knowledge mediators. The way these four concepts relate to each other in product development is similar to the description given by Womack and Jones (2013). So by having cadence in product development for example in the rate of introducing new products can enable flow and eventually lead to pull effect being born. However in order for this to succeed, introducing cadence needs to be done across the business of the company (Oosterwal, 2010). Same applies to flow, too. (Ward & Sobek II, 2014)

This chapter tried to summarize some of the main characteristics of lean product development and how it relates to lean manufacturing. Lean product development is interested in how to run lean product development project and lean product development as a whole. The difference of lean software development theory to lean
product development then is that lean software development is interested more on things related to software development specifically, such as coding and testing.

2.4 Relationship of Lean and Agile

First there was waterfall. Then came agile & iterative development. Now everyone’s talking about lean software development. What’s this all about and how do agile and lean now fit together. Both lean and agile software development obviously have the same goal, which is to produce high quality software, faster and with fewer resources. Simply put the aim of any improvement activity in general is to do things better than they are currently being done. In Agile we talk about things like Scrum and XP when in Lean we talk more about things like optimizing the value chain and eliminating waste. Scrum, for example, can be seen as a method inside agile and agile can be seen as lean thinking put into action. But it is of course not as simple as this. Systems Thinking is a common tool that is used to map out organizations and processes in lean and is explained in more detail in the previous chapter. What is meant by this is that lean is seen as a platform to build agility on. It is something of a wider scope and can be seen more as an enabling factor for agility. One clear view on separating these two is that lean is on the principle level as lean thinking and agile is then on the practice level. (Wang, Conboy, & Cawley, 2012)

Recently the research has also taken a look at how lean and agile are or could be combined in software development (Rodríguez, Partanen, Kuvaja, & Oivo, 2014) (Wang et al., 2012). The usage of lean lean and agile in Finnish software industry has been studied by conducting a survey (Rodríguez, Markkula, Oivo & Turula, 2012). This study points out that lean is mainly used in conjunction with agile and very little only by itself only. It has been studied that these two can be combined meaningfully and without unmanageable conflicts because they, once again, represent different things (Wang et al., 2012). Rodríguez, Markkula, Oivo and Garbajosa (2012) studied the different drivers of combining lean and agile in software development companies. From the results of this research it can be seen that there is power in combining agile and lean but the particular aspects of each organizations strategies should be considered when doing this.

Wang et al. (2012) identified six types of lean application in agile software development. First of these types is the Non-purposeful combination of agile and lean which basically means that these two are combined without knowing that there’s a difference and are pretty much just seen as “the same”. Then there’s Agile within, lean out-reach which means that lean is used to interact with necessary business units but agile is continued to be used as an internal process. Third type of application is Lean facilitating agile adoption when lean is being used to help in adopting agile principles, before or during the process. Lean within agile on the other hand means the usage of lean elements to improve agile software development. This one’s rather interesting and can be seen as a true combined usage of these two. One usual occurrence of this is the usage of Kanban in Scrum teams. Fifth type of lean application in agile software development is called From agile to lean which occurs when lean techniques are used to support agile development and eventually they start to play bigger and bigger role and the end result is the shift from agile to lean where agile methods then take the supporting role. The last type is Synchronizing agile and lean is a special one. In this case inside the organization two teams are working towards the same goal and cooperating but the other one uses agile methods and the other one lean methods. This occurred so that the other team was running traditional agile development mode with
scrum and sprint and the other team was delivering bug fixes and small features by implementing Kanban mode.

Kupiainen, Mäntylä and Itkonen (2015) have done a systematic review of industrial studies on metrics for measuring agile and lean software development. They state that because the difference between agile, lean and Kanban methodologies is not clear, they included all studies in their review with one or several of the three as the focus. The results list various metrics that are being used for measuring lean and agile in software development, of which velocity and effort estimates are the most commonly used.

Based on the discussion in the earlier paragraph it can be said that there are various ways that agile and lean can be combined. These ways might not even be that obvious. Lean can also be seen as a way to scale up agility (Wang et al., 2012). Agile methods might not be enough to really scale agility and this is the case where lean thinking can help. When we start to talk about synchronizing development activities, optimizing the whole value stream and eliminating unnecessary non-value adding activities, wastes, we can rather quickly realize that the things agile software development offers are not enough. Whereas agile is focused on the practical level and project management, lean seems to be the real vehicle to scale up in organizational efficiency. Simply put, it can be also seen that lean tackles different problems than agile and because of this, minimal overlapping between these activities occurs (Wang, 2011). Laanti (2012) mentions that Lean thinking aligns well with Agile software development because they both look at systems holistically instead of reductionism. “This is why lean management principles fit managing agile development better than reductionist management methods” (Laanti, 2012).

2.5 Enterprise Transformation

Organization-wide transformation is what is needed for a company to really change for the better. It is no easy job obviously. Enterprise transformation is further explained by Nightingale and Srinivasan (2011) in their book called Beyond The Lean Revolution. Enterprise transformation paradigm can be seen as an evolution from lean thinking and lean enterprise. What they mean by enterprise transformation is the process of taking a company from its current state to the envisioned future state. This requires changes in mindsets and adapting a holistic view on the enterprise in hand. You really need to know what your enterprise is like. The reasons, for embarking on a journey to enterprise transformation, are many. Maybe the previous improvement projects have yielded benefits on the low level but these benefits have not spread on the enterprise level. Maybe an attempt to transform the company failed from one reason or another. Lean thinking and lean enterprise value are the two things that underlie enterprise transformation. Lean thinking traditionally focuses on eliminating waste and minimizing resources when lean enterprise value aims to recognize stakeholder value. But these two are not enough because they do not provide the holistic enterprise-wide specific tools or methods to really drive the improvement. Transformation requires episodic and continuous change that is aligned. Seven principles of enterprise transformation exist and need to be followed in order to succeed in transforming an enterprise. These principles are deeply connected to each other and do not yield the desired benefits if focused on in isolation. (Nightingale & Srinivasan, 2011)

Nightingale and Srinivasan (2011) present seven principles of enterprise transformation. These principles are 1) Adopt a Holistic Approach to Enterprise Transformation, 2) Secure Leadership Commitment to Drive and Institutionalize Enterprise Behaviors, 3) Identify Relevant Stakeholders and Determine Their Value Propositions, 4) Focus on
Enterprise Effectiveness Before Efficiency, 5) Address Internal and External Enterprise Interdependencies, 6) Ensure Stability and Flow Within and Across the Enterprise and 7) Emphasize Organizational Learning. By following these principles, organizations have guidance on their transformation journey.

The Enterprise Transformation Roadmap can be used to guide enterprises through their transformation journey. It was created to guide companies to successful transformation journeys because most enterprise transformations fail for various reasons. One type of an enterprise transformation failure is Flavor-of-the-month, which means that the enterprise undertakes transformation efforts that keep on shifting from one methodology to the other, hence the name. Another type of failure is called New Leadership which means that the transformation heads towards a direction set by an upcoming leader, without considering the current or past transformation initiatives. Both of these examples demonstrate the lack of long-term thinking in the transformation initiatives. The LAI Enterprise Transformation Roadmap introduces three cycles that follow each other in a cyclic manner. These cycles are called: Strategic Cycle, Planning Cycle and Execution Cycle. Strategic cycle is the starting point for any enterprise transformation because there the business case for the upcoming transformation is calculated and the engagement of leadership in this effort is secured. The planning cycle begins with the assessment of the current state of the enterprise, envisioning the future state and creating a transformation plan to be followed in order to fill in the gap between the current and the desired future state. A tool, such as The Lean Enterprise Self-assessment Tool (LESAT) can be used to implement the assessment. This tool has been designed specifically to support the LAI Lean Enterprise Transformation Roadmap. The tool has also been mapped to software development domain (Karvonen, Rodriguez, Kuvaja, Mikkonen, & Oivo, 2012). The final cycle, the execution cycle, according to its name executes the plan. Short-term corrective actions can take place during the execution cycle and long-term corrective action is then updated to the transformation plan in the planning cycle.

The specific challenges of ICT companies in lean transformation have been studied by Suomalainen, Kuusela, Teppola, and Huomo (2015). In their study they use the framework proposed by Kuusela and Koivuluoma (2011), who adapted the LAI MIT Enterprise Transformation Roadmap for software intensive companies. They’ve reported the challenges faced by software intensive enterprises when transforming to lean. It was emphasized that companies should use a transformation roadmap when embarking on their transformation journeys. Another observation was that more money should be used to tackle soft-issues with people such as learning. On the other hand those hard-issues like organizational tools and systems are much easier to change with moderate investments (Suomalainen et al., 2015).

The differences in the scope between this study and the studies by Suomalainen et al. (2015) and Nightingale and Srinivasan (2011) are clear. Where the aforementioned two focus on transforming on a high organizational level, this study focuses on finding improvements on a lower level by using lean principles in the specific area of software development.

2.6 Lean Software Development

Lean software development is something that wasn’t born separately from lean manufacturing and lean product development. It has its roots in the same principles drawing back all the way to Toyota and Taiichi Ohno. Because software development differs greatly from manufacturing or designing cars, it is obvious that the same
principles can’t be applied off-the-shelf. This is why separate, adapted guidance is required to apply lean to software development. Lean software development has been recently studied from various point of views. This chapter lists current research on these various approaches to lean software development.

Rodriguez et al. (2013) studied the application of lean thinking in the context of a large-scale telecom software development organization, Ericsson AB. The underlying theory in their study was based on the original five principles of lean and they used those for mapping the lean software development characteristics of the company. They found challenges and limitations for applying lean to software development in this specific case. Achieving flow and transparency and creating a culture of continuous learning were seen as the biggest challenges. On the other hand creating a culture of continuous improvement, involving people in the transformation and creating a team culture were seen as more easily achievable elements of lean.

Another case study at Ericsson AB was also conducted but with a lot smaller scope by Antinyan et al. (2014). They studied the possibilities of identifying risky areas of software code in an agile and / or lean software development environment. They present how assessment of risks in this environment can be used to manage those risks. Yet another case study was conducted also at Ericsson AB by Karvonen, Rodriguez, Kuvaja, Mikkonen and Oivo (2012) where LESAT was adapted to software development and then assessed. From these three studies mentioned above it can already be seen that lean can be studied on various levels in software development from the risky software code to valuable principles and enterprise transformation.

Ebert, Abrahamsson and Oza (2012) state that lean software development attempts to fill the gap that is created when only specific improvement methods are introduced with short term goals. It is seen that lean in software development is still interpreted in various ways so that people just end up fighting about methods. To further study what lean in software development really is about, empirical studies must be done in industries (Ebert et al., 2012). It can be seen here that Rodriguez et al. (2013) have done just that, they’ve taken the five original lean principles and used them as their backbone theory in a case study.

Value is the first and also most central principles in lean (Womack & Jones, 2013). Providing value to customers in software development by using lean principles has been studied (Mehta, Anderson & Raffo, 2008). In this study they tried to tackle several software development problems by using lean principles. Things such as the importance of team culture, management involvement, proper planning stop-the-line principle, improvement of flow and introducing continuous integration were presented. They also see Kanban as an effective method to limit work in progress (WIP). The importance of Kanban as a lean software development method to drive team orientation, learning and communication and collaboration has been recognized by Oza, Kettunen, Abrahamsson and Münk (2011). Mehta et al. (2008) eventually state that lean is about removing all activities that do not add value for the customer. This is in accordance with the original lean theory (Womack & Jones, 2013). Solutions to problems in software product management by using lean principles has been studied similarly to Mehta et al. (2008) by Maglyas, Nikula and Smolander (2012). Some of the solutions they provided were the usage of flow to decrease time to market, using value to determine key performance indicators (KPI) and using perfection for incremental changes. Here it can be seen that software product management is yet another level on which lean software development can be studied.
Mehta et al. (2008) mention that to achieve flow in software development, one way is to go towards delivering just in time (JIT). This means that the customer pulls work from the supplier instead of the completed work standing in queues or worse, being pushed to the customer. Petersen and Wohlin (2010) have studied the measurement of flow in software development further. They demonstrate how flow in software development can be visualized to reach the goals of increased throughput and reduced lead-time. They also proved that having a system to show progress of software product development is seen as a useful thing from the practitioners’ point of view to support the concept of achieving flow. The management of flow has also been studied from the software architecture point of view (Nord, Ozkaya & Sangwan, 2012). In this study it is said that “By visualizing architecture-related tasks that contribute to feature throughput, including tasks that span multiple sprints, teams can achieve and effective flow-based development environment” (Nord et al., 2012). They suggest WIP limits, Kanban and monitoring of the quality of the system as important aspects when trying to achieve flow in software development.

It can be already seen that lean software development can be mapped from various points of view. Lane, Fitzgerald and Ågerfalk (2012) present a candidate list of lean principles for software development synthesized from different sources where lean thinking has been applied. This list contains twelve values, such as continuous improvement, customer value, effective collaboration, flow of value and remove waste. These values are linked to the relevant sources. A similar study has been done by Jonsson, Larsson and Punnekkat (2013) where key lean concepts for software development are first derived from the literature. Based on this they suggest a framework with a set of goals, recommended activities and practices. The four main categories for lean software development concepts listed are main philosophy, process, people and technology (Jonsson et al., 2013). A systematic review of approaches to large-scale software systems development has been conducted also (Pernstål, Feldt & Gorschek, 2013). This systematic mapping study concludes that most of the results in studies focus on creating flow and eliminating waste in software development context. This study identifies that there is a need for further research in studying the application of lean principles in software development, especially in large-scale companies. The study also states that there is a gap in the research about the application of lean product development (LPD) principles in software development.

Software project management is another aspect of applying lean to software development and has been studied by Middleton and Joyce (2010). In this study the power of lean methods such as visual management, smaller batch sizes, team-based process control and statistical process control was proven. The application of these approaches resulted in the improvement of lead time, consistency of delivery and the occurrence of defects. Difference of lean and agile is also highlighted by showing how lean has a wider scope and promotes long term improvements. One of the key challenges they identify is the problem of fitting this new way of lean thinking into the organizations existing structure.

It can be seen from the existing research that lean software development can be observed and implemented in many ways. Poppendieck and Poppendieck (2003, 2006) present the seven principles of lean software development that contain the underlying concepts in almost all of the research presented in this chapter. These principles will be presented in more detail in the following chapter.
2.7 Seven Principles of Lean Software Development

In their book *Lean Software Development: An Agile Toolkit*, Poppendieck and Poppendieck (2003) present seven principles for driving Lean Software Development. These principles were later renamed slightly, when their book *Implementing Lean Software Development: From Concept to Cash* was published in 2006. These principles are 1) Eliminate Waste; 2) Build Quality In; 3) Create Knowledge; 4) Defer Commitment; 5) Deliver Fast; 6) Respect People; 7) Optimize the Whole. The principles themselves are just that, principles. They do not provide off-the-shelf solutions for efficient and optimized software development but rather they present different aspects that should be considered when making improvements. By following the principles and their ideas one guarantees that no vital area of focus gets forgotten in the process of improving. These principles will now be presented in more detail.

2.7.1 Create Knowledge

As software development is largely considered as knowledge work it means that knowledge is created throughout the lifecycle of a project. Knowledge is created through experience and is distributed in organizations and its projects in various ways from systematic wikis to coffee room talk between employees. The first underlying aspect that arises from this principle is learning based on experimenting and the knowledge created through this (Poppendieck & Poppendieck, 2006). This simply means that failure is always a learning opportunity because it creates the knowledge about how that certain thing should not or can’t be done. This can be achieved by pursuing set-based development in which the options are kept open by proceeding in the development process with a set of possible solutions. This results in learning about the possibilities of these solutions and in this way creating usable knowledge. This knowledge can then be used for making scientific decisions on which of the solutions is the best one to pursue all the way to the actual product development (Ward & Sobek II, 2014).

Three different types of learning can create usable knowledge. These types of learning are 1) Integration Learning which occurs when we learn about the environment in which the product will be used, referring to customers, suppliers and such, 2) Innovation Learning which happens when new possible solutions are created and 3) Feasibility Learning which happens when people understand better decisions that can be made within the existing solutions (Ward & II, 2014). Although these types of learning have been recognized in the context of Lean Product Development, one only needs to use a bit of common sense to understand that they adapt to software development domain, too. In lean product development, practitioners often create so called trade-off curves and limit curves to present the usable knowledge in a more understandable format. These curves usually present the technical limits in which a technical solution can be made and the trade-offs that happen when optimizing a certain variable. The applicability of this concept to software development can and should be challenged.

Finding the root cause is also something very central to lean thinking and can be seen to create knowledge. The principle of The Five Why’s originated from Toyota and it simply means that every time a factory worker at Toyota encountered a problem they had to ask five “why” questions (Womack et al., 2008). The results of this process should be the discovery of the root cause for the problem. Then appropriate action is taken to fix this problem and ensuring that the same cause does not cause the problem again. “Stop-the-Line” principle is also usually the initiator for this problem solving process because according to this principle every time a factory worker encountered a
problem they would pull a cord called the Andon Cord. This made occurring problem in the factory visible to everyone and especially for the people who needed to take action. Later on this principle has been expanded to mean all types of visual message boards that can be easily modified to present important information. In software development domain root cause analysis / “The Five Why’s” principle can then be used to avoid technical debt for example by tackling problems in the code base early on and right when they occur by refactoring. In this way it can be seen that the responsibility of improving development process lies in the hands of people involved in it (Poppendieck & Poppendieck, 2006)

2.7.2 Deliver Fast

A common misconception in the industry still exists that one cannot deliver rapidly, while maintaining high quality and low cost. With modern test automation and continuous integration systems, the increase in delivery speed of software should not automatically be the reason for increased number of defects. Increasing the speed of software delivery does not really mean one programmer doing the same amount of work in half the time. Instead it means organizing the whole delivery pipeline in such a manner that no unnecessary waiting occurs. In other words this means creating flow to the delivery pipeline, a direct reference to lean thinking. In order to do this, shorter cycle times should be driven in the development as well as small batch size. This means that the progress of delivery is also more easily tracked. Queuing theory applies here also (Poppendieck & Poppendieck, 2006). The challenge in delivering rapidly then becomes the work of organizing various development activities in such way that they get completed at the same time if there are dependencies and no one has to wait for anyone. If code is developed and deployed and then left unused, it is waste from lean point of view. Small batches also reduce dependencies so problems can be more easily isolated and tackled. Building small blocks of working functionality on top of each other is the way to go. (Poppendieck & Poppendieck, 2006)

There’s more value to the act of delivering fast than one might think. In 2015 software companies are faced with more and more demand from their customers with ever changing requirements. When a company starts to implement the principle of delivering fast one unexpected benefit they might face is the fact that the customer has less time to change their mind because software is being delivered continuously and functionality demonstrated all the time. This gives more control to the company doing the development work and reduces the amount of rework after changing requirements.

For software to be able to be delivered in a constant flow, the work done needs to be limited to capacity. This increases predictability. If work would not be limited to capacity, predicting development times would become harder than it already is in the context of software development. This would further then reduce the possibility to create flow in to the delivery and actually slow it down. The key is to manage workflows instead of schedules. (Poppendieck & Poppendieck, 2006)

2.7.3 Respect People

Maintaining co-located, cross-functional and self-organizing teams is a key to successful software development. When small teams are working together on a piece of software it yields all kinds of benefits. For starters the knowledge-transfer between the developers is easier. This is how scrum teams work. The whole team consists of people having competence from various functional areas: testing, programming and specification work, for example. Because of this shared competence the team is able to
work rather autonomously and rarely needs consultation from outside. (Poppendieck & Poppendieck, 2006)

It is not enough to have a competent group of people working towards a common goal if there’s no one in the team with leadership skills. Likewise if there’s no captain on a ship, the ship is likely to sink. At Toyota this person is called the Chief Engineer. Usually what is meant by a Chief Engineer is a person responsible for the business success of a vehicle family (Poppendieck & Poppendieck, 2006). But the concept of Chief Engineer can be brought couple of levels down to mean a person in a team that has understanding on several functional areas of the product and at the same time also understand the business side of it.

One aspect of respecting people is to bring decisions-making to the lowest level where it is possible to be made (Poppendieck & Poppendieck, 2006). This is the beginning for making sure that right decisions are made. If decisions that are needed on level 5 are made all the way up at level 2, common sense tells you that something is wrong in that picture and that those decisions will most likely turn out to be faulty. Scrum teams are a great example of empowering people on the lowest level to be able to make decisions about things that affect them directly. This way of working also contributes to increased speed in the development work and tears down the structures of bureaucracy.

2.7.4 Defer Commitment

By Defer Commitment principle we mean getting rid of the idea that product development should start with a complete and final specification set. People learn by doing and it is rarely possible to know the best solution in the start. Add changing customer requirements and a dynamic business environment of today to the mix and you start to realize why it is not a good idea to commit to something 100% in the beginning. As a golden rule all irreversible decisions should be scheduled in the last responsible moment. And extending this thinking even further, the starting point should be the removal of irreversible decisions and replacement of those decisions with reversible decisions. This is of course not possible for all decisions because in each development effort there are points when irreversible decisions have to be made. Maintaining your options there where change is expected is important. When a software developer is developing code, he / she should always be keeping options open up to the point that it is meaningful. It might be that keeping too many options open when programming results in generic classes covering everything but actually nothing. Taking this up a notch, removing dependencies in the architecture level can yield more extensive gains. Software is initially developed according to the architectural plans and those plans create the base on which to build the actual application or system. Dependencies should be straightforward and transparent, so that no hidden dependencies exist. Also it should be remembered that software architecture is initially done before actual development work starts but the actual programming then defines that how well the architecture gets implemented. This means that even the most perfect architectural plan can be useless if not implemented according to instructions. This applies from building houses to building software. (Poppendieck & Poppendieck, 2006)

One additional note is that planning should not be synonym for commitment. When planning, you should always do it so that you do not create too many points in to the plan that result as irreversible decisions. Instead you should always be doing planning so that flexibility is maintained and the plan can be modified on the way towards the goal. (Poppendieck & Poppendieck, 2006) Planning is necessary, commitment is not.
The attitude should not be that tough decisions are done in the early phase to have them “done and dealt with”. This kind of decision making is usually done with insufficient knowledge, which leads to problems and failure. Committing early on can create a false sense of security and usually only later in the process it is revealed that work has been done based on a decision, with nothing to support it. For tough decisions you should instead be experimenting with various solutions and gaining great understanding through these efforts to be able to decide with knowledge and confidence.

### 2.7.5 Eliminate Waste

The original seven types of waste defined by Womack and Jones (2013) have been mapped from manufacturing to software development (Poppendieck & Poppendieck, 2003). As mentioned in the earlier chapters, *muda* is the Japanese word for waste and it means any activity that does not add value to the product. Eliminating waste respectively means identifying and then removing all these unnecessary non-value-adding activities from the products value chain. Of course the underlying reason for eliminating waste is to increase efficiency and so on to create better products with fewer resources. It must also be remembered that the different types of waste are only listed separately to help people understand the wide-spread of the possible non-value-adding activities. The effect of removing one category waste is always determined case by case, organization by organization.

*Partially Done Work* is the first type of waste that is listed here. In manufacturing this waste is called In-Process Inventory. Examples of Partially Done Work in software development are Untested Code, Undeployed Code and Undocumented Code. In other words when work gets left “hanging”, it is not possible to track the progress and incomplete data gets distributed.

*Extra Features* in software development are waste and correspond to over-production in manufacturing environment. Simply put, if there is not a clear demand from customer for a certain feature in the backlog, this feature should not exist at that time and should not be implemented into the system. The Pareto Principle: “80 percent of effect come from 20% of causes” has been applied here as “20 percent of the features create 80 percent of the value” (Poppendieck & Poppendieck, 2006).

*Relearning* is what happens when you by accident or without the proper knowledge try something that has been already done. In manufacturing this waste is referred to as extra processing. The challenge here is how to manage all the created knowledge in a way that the people working would actually study and use it. A systematic and rigorous way of collecting and distributing created knowledge should be established. This is something that the field of Knowledge Management is studying and a lot of research has already been done as pointed out by a systematic review on knowledge management in software engineering (Bjørnson & Dingsøyr, 2008). In addition to knowledge management systems, emphasis should be put on involving all relevant people in the development process to create natural distribution of knowledge and decrease the amount of relearning taking place. (Poppendieck & Poppendieck, 2006)

*Handoffs* happen when a work gets passed from one actor to another. In this event the work moves from the first person to the next and what usually happens is a loss of knowledge in between. Work usually contains so much knowledge that it all can’t possibly be documented. So, when handoffs take place, this knowledge gets lost and as a result the person continuing the work will most likely have problems to do their job properly because of the lack of knowledge. Some ways to tackle the negative effects
that the handoffs create are: Reducing the total number of handoffs, Usage of cross-functional teams where knowledge is kept within and problems in handing work off to your colleague can be more easily solved and replace heavy documentation with Q&A sessions, other face-to-face meetings and visual presentations. In manufacturing context handoffs occur as transportation. (Poppendieck & Poppendieck, 2006)

*Task Switching* means just that, switching work tasks. Most of the tasks in software engineering projects are knowledge work and require a lot of concentration and thinking so if people doing work tasks keep on switching from one to the next constantly, none of the tasks actually progress and the result is that no work gets done. Another scenario is where people try to do several tasks at the same time, also known as multi-tasking. In this scenario the exact same thing usually happens. Programmers can be for example switching between maintaining old legacy codebase and at the same time developing new code for a different project. Another example is a developer that is developing new features but time to time, unexpectedly, needs to react to hot fixes for found bugs and jump off from developing the new feature. These can be tackled by assigning certain people at a time to take care of the maintenance of the old code base. This would allow others in the team to focus on certain tasks and finish up with them, after which a rotation could be possible. (Poppendieck & Poppendieck, 2006)

*Delays* in software development occur when a developer for example has to wait another developer to be available in order to make decision. It often happens that decision to be made requires the expertise of someone else and if this person is not available it causes delay. Or in a more general level, a delay occurs when the required knowledge is not available in the right place at the right time for a blocker to be solved. Challenge then becomes of having just the right knowledge in just the right place at just the right time. This is what we mean by Just-in-Time (JIT) principle. In manufacturing this waste is referred to as *waiting*. (Poppendieck & Poppendieck, 2006)

*Defects* in software development occur when the software or the system fails to meet the requirements in one way or another. Defects cause rework, decrease customers’ satisfaction and cost time just to name a few. The ultimate penalty being loss of money, as it often is. To prevent defects from occurring, software testing takes place. Embedding automated unit-tests to the code base to create safe “scaffolding” for future development is just one of the many good testing practices there exists. This makes the code more robust and maintainable. Also the principles of testing early in the development should be followed so that the possible defects can be spotted right there and then. Zero defects in the delivered product should be the goal for all testing strategies. Defect is also the term used in manufacturing to describe this waste. (Poppendieck & Poppendieck, 2006)

### 2.7.6 Build Quality In

Building quality in to the software right from the start seems like an obvious thing to do but it is easier said than done. What is meant by building quality in is that you want to avoid all kinds of defects right from the start. This can be done by creating a proper test strategy and implementing it. To create quality you should always be inspecting for preventing defects and not after they occur. Unit-testing is automated testing at the lowest level. Of course you then have different levels of testing from there on up, such as manual free testing, automated stability testing, performance testing, user acceptance testing; you name it! This kind of process for creating software can be called mistake-proof process. It is further appraised when continuous integration systems are taken into use and developed code fragments or modules are continuously integrated to the
codebase and it happens, automated integration tests are ran to guarantee the quality of the newly produced piece of code and its fit to the existing code. Also one separate factor is architectural dependencies in the developed software. When you break dependencies on the architecture level, your code becomes more flexible for late changes, improvements and maintenance. Fully unit-tested code base then should guarantee that if and when a dependency breaks, this is alerted and the defects or the technical debt lurking below the surface is revealed. (Poppendieck & Poppendieck, 2003, 2006)

2.7.7 Optimize the Whole

The principle of optimizing the whole, this is what lean is about. Instead of focusing your improvement activities on isolated areas, one should be looking at the organization and its processes as a whole. If you optimize in isolation you only end up with huge amounts of sub-optimizations with no benefits for the end outcome. The whole pipeline needs to work together, not only parts of it. From a project point of view this needs to happen from the customer request all the way to the phase when working software has been deployed to the customer. (Poppendieck & Poppendieck, 2006)

One way organizations and their processes can be approached with the “optimizing the whole” mindset, is Systems Thinking. It is used to solve systematic and complex problems in organizations. It is the opposite way of thinking to sub-optimizing.

The mindset in software companies should be to deliver a complete product, not just the software. This again brings us to the idea of optimizing the whole. If you for example leave a good customer support out of your product and only deliver the software to the customer, that same customer is highly unlikely to be contacting you again for an order. Another aspect to delivering a complete product instead of software is companies delivering embedded systems. These companies need to integrate working software with working hardware and as a result we have a system of high complexity. In this kind of companies focusing only on improving software development processes or only on improving hardware development processes just is not enough. Instead you need to map out the interaction that goes on between various activities that make up the end product. Of course software and hardware development activities are only a part of processes needed to create, deliver and maintain an embedded software product. The key message here is to align and synchronize your activities in the most efficient way so that flow is maintained throughout the products value stream.

*Lean software development theory by Poppendieck and Poppendieck (2006) was selected as the core theory for this thesis and is used as the underlying theory for running the focus group sessions. The selection was done based on the largest number of references under the topic of lean software development in primary papers and in Google Scholar as identified by (Jonsson et al., 2013). Another factor affecting the selection over other earlier mentioned lean software development theories, is the fact that the theory by Poppendieck and Poppendieck (2006) is very well explained so that it is easier for practitioners to understand in a limited time. This means that the theory presented in this chapter is used to trigger discussion and reflect back to when analyzing the collected data. In other words, this chapter presents the frame where this research sits in.*
3. **Research methods**

The research in this thesis was conducted as a case study and data was collected by organizing focus group sessions. The research is qualitative. This chapter will explain theory about the case study method, qualitative research and the focus group method. Then the design of the research will be explained. Finally, it will be explained how the research was done in practice.

3.1 **Research Questions**

This thesis aims to provide answers to three separate research questions. The first research question is: "What is the practitioners’ understanding of lean software development?". Answering this questions helps is assessing the interpretation of lean software development by people working in the industry. As mentioned in chapter 2 about the fuzzy line between agile and lean software development, it is expected that these two will be mixed up to some extent.

The second research question asks that: “What are the opportunities for applying lean software development?”. Respectively, the third and final research question then asks: “What are the barriers for applying lean software development?”. These two questions complement each other because when we want to know opportunities in implementing something, it is also relevant to know possible barriers for implementing it. Rodriguez et al. (2013) also investigate similar research questions as the last two in this research. The questions they investigate ask that: “What elements of lean thinking are challenging the implementation of lean in software development and what are the elements of lean thinking that are more easily achievable in a software development context?”. The case in their study is also similar to the one in this research, which is a large-scale software intensive company. This enables the possibility to reflect the results of this research with the findings by Rodríguez et al. (2013). Another reason for taking a look at the opportunities and barriers in lean software development is that it really is quite a natural thing that there’s a need to know about the opportunities of doing something and what are the worst barriers of doing something. This keeps the balance because we are looking at both aspects; good and bad at the same time. Barrier refers more to a thing that needs to be taken down before applying something can be done. It can also be seen that removing a barrier can usually open up an opportunity or many.

3.2 **Qualitative research**

The most usual distinction between different research methods is to divide them into two: quantitative and qualitative research. Quantitative research has its roots in natural sciences and examples of it are laboratory experiments and survey methods. As the name says, quantitative research is interested in numerical research data. Qualitative research has its roots in social sciences and it was developed to help us understand how people behave in different cultural or social context and what these contexts are. Case studies and action research are examples of qualitative research. Also ethnographic research and grounded theory research are qualitative research methods. Data sources for qualitative research can be observation, interviews, focus groups, archival data and
researcher’s impressions. The research in this thesis is *qualitative*. (Myers & Avison, 2002)

There can be three underlying philosophical assumptions in qualitative research. These are *Positivist, Interpretive* and *Critical*. Positivists think that the reality can be observed objectively and usually relies on theory building. Interpretive view places emphasis on the importance of language and social constructs as ways to access the reality and sees that understanding meanings of these is important. Critical researchers then again highlight the circumstances where in people act so that these circumstances can restrict the people’s ability to act in a particular way and that these conflicts and oppositions are something to look at when studying the social environment and the people in it. (Myers & Avison, 2002)

The importance of qualitative research methods in empirical software engineering was recognized already in the 80’s (Curtis, Krasner & Iscoe, 1988). In the late 90’s Seaman (1999) also studied the topic. There are several advantages in the usage of qualitative research methods over quantitative in software engineering research as listed by Dybå, Prikladnicki, Rönkkö, Seaman and Sillito (2011). Firstly quantitative research can be tough to conduct because of potentially small sample sizes and expenses in conducting controlled experiments. Qualitative studies can also offer grounded hypotheses and results that are able to handle the complexity of the topics. When variables are not defined explicitly, this is also when the qualitative methods step in. But above all these, the biggest advantage that the usage of qualitative research brings, is it forces the researcher to dive deep into the complexity of the studied area and to really look deep on the matter. In software engineering especially when so much of what happens is about people, their relations and the environment they act in, qualitative research can yield great results in understanding various phenomena (Dybå et al., 2011).

### 3.3 Case Study Research

The suitability of case study research is highlighted by Runeson and Höst (2008) in their publication where they present guidelines for conducting and reporting a case study in software engineering domain. The main reason because it is seen as a suitable research methodology is because it studies contemporary phenomena in its natural context. Software engineering research area can be compared partly to the areas of psychology, business and sociology and because the methodology is popular amongst these areas, it can be seen suitable for software engineering too. The nature of software engineering involves different stakeholders developing and operating in ever changing environments. Because of this, many of the research questions rising in this domain can be tackled by conducting case studies. Because the boundary between the phenomenon and its context is often unclear in software engineering, a case study is even more suitable research methodology for this domain. (Runeson & Höst, 2008)

Case study research is one of the many qualitative research methods and is said to be the most common one of them (Myers & Avison, 2002). According to Yin (2009) the definition of case study research methodology is twofold. Firstly we define the scope of a case study: “A case study is an empirical inquiry that investigates a contemporary phenomenon in depth and within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin, 2009). This statement gives a clear distinction between case studies and, for example, experiments. Whereas case study tries to study a phenomenon in its real-life context, experiments require the control of behavioral elements (Yin, 2009). In other words, this means that control is needed whereas in case studies this control is given away to let the natural behavior and
environment to flourish. The control in experiments is typically a result from the so-called “laboratory environment” (Yin, 2009). The second part of the twofold definition for a case study handles the fact that because it is not always possible to distinguish phenomenon and context in a real-life situation, data collection and data analysis as other technical characteristics need to be accounted. The definition continues as follows: “The case study inquiry copes with the technically distinctive situation in which there will be many more variables of interest than data points” and “As one results relies on multiple sources of evidence, with data needing to converge in a triangulating fashion” and “As another result benefits from the prior development of theoretical propositions to guide data collection and analysis” (Yin, 2009). This definition presents all the important aspects of a research method, consisting of the logic for design, data collection techniques and approaches to analyze the data. This means that case study is not just a tool for collecting data but a full research method in itself. (Yin, 2009)

The case study research process is a linear but iterative process, consisting of Planning, Designing, Preparing, Collecting, Analyzing and Sharing. These phases and their meaning will be briefly explained next. Figure 3 gives a nice overview of these phases and their relations to each other. In planning phase the decision to use the case study method as a research method is made and this starts the process. The researcher must study the method to fully understand its suitability for the study. Also a key part of the planning phase is to identify the relevant research questions and these research questions can be used to justify the selection of the case study research method. (Yin, 2009)

![Figure 3. Case Study Research Process, redrawn based on Yin (2009).](image)

In the design phase the design itself is identified, whether it is a single, multiple, holistic or an embedded case study. The main difference between these is determined by whether the case is studied as a whole or if there are multiple units of analysis inside a
case. It is also possible that multiple separate cases are studied. The selection of the case and the possible units of analysis have an effect on identifying the design and vice versa because the design can be decided before selecting the cases and units of analysis. Theory, propositions and issues related to the study are determined and needed procedures for ensuring the quality of the study are recognized. (Yin, 2009)

The third step in the case study research process is preparing for the study. This includes ensuring the proper skills needed to investigate the study, possible training for a specific case, development of a rigorous protocol for the case study and conducting a pilot case. (Yin, 2009) One of the most important skills for a case researcher is the ability to stay objective and not let personal opinions affect the outcome of the case. It must be remembered that the investigator is observing the case, not interfering with it too much. One other important aspect of the prepare phase is to gain approval for human subjects protection so that the researchers is able to keep the privacy of the studied human subjects. For example formal consents might be needed. From all of these mentioned points of the prepare phase, following the case study protocol is one of the most important ones. The case study protocol consists of an overview of the case study project, field procedures such as access to the case study sites, case study questions that the researcher needs to keep in mind and a guide for reporting the case study including an outline for the report. (Yin, 2009)

According to its name, the collect phase is where the case study data is eventually collected. The case study protocol is followed, data is collected from multiple sources of evidence, case study database is established and chain of evidence is maintained at all times. Maintaining a transparent path from the data source to the collected data, and eventually to analyzed data and from there to the research report, is extremely important. This contributes to the validity of the study because anyone should be able to track observations all the way to the interpreted results. Different sources of evidence that can be used to collect data from in this phase are for example documentation, interviews, archival data and direct observations. Using multiple sources of evidence contributes to the validity of the study. Interview is seen as one of the most important tools of case study information (Yin, 2009)

Analyze phase is the point where the data collected in the previous phase gets put in to use. The data needs to be presented so that it can be separated from the interpretations. A general strategy for analyzing the data helps in creating a structure for it. If this is missing then the researcher might need to “play around” with the data to see what kind of structure the analysis might take. Then the five different analytic techniques need to be explored and the most suitable for the study should be selected or at least used as a reference and help for analysis. These five techniques are pattern matching, explanation building, time-series analysis, logic models and cross-case synthesis. These approaches can be used, as mentioned previously, as reference for building your own analytic strategy and they do not provide silver bullet strategy for the analysis. One important aspect is also to always explore existing “competing” explanations of the same phenomenon and do analysis based on this. When analyzing, one should be reflecting with up-to-date research literature in order to achieve rigorous analysis. (Yin, 2009)

The final phase of the six phase process for conducting case studies is naturally the sharing phase. In this phase the results of the study are shared to the audience. In order to be able to share relevant results in the most meaningful way, this phase starts by defining the target audience. In the academic research environment, the audience is naturally the academic researchers and students. But case studies might have a wider
audience, consisting of example practitioners from the industry or funders of the research. These differences need to be accounted for and the reporting should be done to serve these audiences, respectively. The aim is to make the audience understand what you want to say. Communicating the results, however, is more easily said than done. Some statements exist that define an exemplary case study. Firstly the case study must be significant so that it holds actual value. The case study must be “complete” in a sense that it considers all relevant aspects and is conducted holistically. The reader should be able to “sense” the “completeness” of the case study from the report. The case study must also consider alternative and competing perspectives. It is not the correct way to do research if only one perspective is explored and considered when reporting a case study and hence, various other perspectives should be objectively considered also. Providing sufficient evidence for the reader to make their own interpretations is needed so that the author’s view on the matter is not needed to be trusted blindly. And last but not least, the case study report must be engaging which means that the report flows in a sensible manner so that the reader maintains interest and continues reading. In a way, the author is tempting the reader to continue reading to the next sentence, the next page and to the next chapter. The process of writing the report is iterative and re-writing is the key to successfully reporting a case study. (Yin, 2009)

To sum up the case study research process, the relations of the phases discussed above can be observed from figure 3. It all starts with planning the case study after which the researcher proceeds to the design phase. When the design is done, the researcher prepares for the case study and then collects the data. It is possible, and also recommended, that after the first round of data collection and analysis; the design is enhanced for example by improving the way the data is gathered. Eventually when enough data has been gathered and analyzed, the results and the preparation for the case study are shared to one or several audiences in the form of a case study report. (Yin, 2009)

3.4 The Focus Group Method

The focus group method is a qualitative research tool (Langford & McDonagh, 2003). Langford and McDonagh (2003) state that: “A focus group is a carefully planned discussion, designed to obtain the perceptions of the group members on a defined area of interest”. The participants are selected based on their personal characteristics in a way, that they support the topic of the session in order for the session to yield the best results. In a focus group session, the number of participants is typically anything from three to twelve and lasts from two to three hours (Kontio, Lehtola, Bragge, & Box, 2004). The sessions are guided by a moderator who is responsible of initiating the discussion, probing deeper as needed and taking care of general facilitation matters. Because the sessions consist of a group of people pondering the topics together, it enables them to build on ideas of others and this creates all more meaningful discussions. The four steps for a successful focus group session are presented in figure 4. The road towards a completed focus group starts by defining the research problems that are needed to be studied. The research problems then determine the suitability and selection of the focus group method and guide the planning of the event itself. After the event is planned, participants are selected based on their characteristics. After the participants are selected, it is time to run the session itself and collect the data for analysis.
The main difference between a focus group session and a group interview is the fact that in a group interview the participants are supposed to simply answer the interviewers’ questions while in focus group sessions the emphasis is on the interaction between the participants and also partly, the moderator. Even though this kind of interaction can occur in group interviews to some extent, in focus group sessions this is the starting point and because of it they end up being completely different. In a nutshell; whereas interviews are done on one-to-one basis, the focus group is a collective way of interviewing and including the participants in the discussion (Langford & McDonagh, 2003). This is one big advantage of the tool.

Kontio et al. (2004) have studied the usage of the focus group method in software engineering. They recognized a list of strengths and weaknesses of the method. The first one of the strengths is the possibility for the participants to build on top of the ideas of others. Related to this the participants are also able to confirm and agree with the opinions of others. The method was also found to be a cost-efficient way of obtaining practitioner experiences. Because of the focused nature of this method, it is also possible to really go deep into the reasons why participants think in a certain way. The last one of the strengths mentioned by Kontio et al. (2004) is the business benefits for participants because they were able to benchmark their thinking and ideas against others and found value in it but also they were able to network and relate with each other during the session. But when there are strengths, there are always weaknesses. The dynamics of the group can have an effect to the activity of discussion and also it can affect the possibility of the moderator to have full control over the flow of the session. This can be tackled by good preparation and a structure for the session. Social acceptability can be a problem if participants get embarrassed when providing answers that are proven wrong by others. Here again, the role of the moderator is important by working as a diplomat. Hidden agendas can have an effect on the focus group sessions in a negative way if participants come prepared to drive pre-thought opinions for various reasons, such as business reasons. This can occur especially when there would be business relationships between the participants and can be obviously limited by selecting participants in a way that this kind of relationships do not exist. Secrecy can occur because of the same reasons as mentioned before and can be addressed by the same means. The last, but not the least, of the weaknesses of the focus group method is the limited comprehension. Because the time is limited in a focus group sessions, a too complex topic might result in fragmented discussion with no real results. Because of this the topic, the experience of the participants and the time available must be considered carefully in order to achieve the best results. (Kontio et al., 2004)

The focus group sessions need to be well managed so that the time is sufficient to cover all the topics required. The session starts by an introduction to the topics given by the moderator, stating the ground rules and answering any questions that may rise. After this everyone introduces themselves after which the sessions get on its way. Many different tools can be used during a focus group sessions to mediate the discussion. Examples of these tools are affinity grouping technique or for example voting. The
usage of this kind of tools to freshen up the discussion is encouraged so that the participants do not get bored. During the sessions, the research data can be captured in many ways. For example audio recording is one of the most popular techniques. Also, simply taking notes can be a good way to get an overview; however it should be noted that working as a moderator is a full-time job so an assistant is required for note taking. When the session ends, the notes written by the assistant can be used to quickly summarize and recap the session. This is important because it makes sure that the participants’ discussion has been understood correctly and any mistakes can be fixed right there at the spot. To analyze the data gathered from a focus group sessions, any combination of the qualitative data analysis tools and techniques can be used. (Kontio et al., 2004)

If something should be most emphasized in running a focus group session, it is the role of the moderator. Most of the weaknesses that have been recognized in running focus groups by Kontio et al. (2004) can be tackled and minimized by a good moderator who is able to make the participants feel comfortable during the sessions, is able to react quickly to problem situations and can improvise when needed. A strong moderator with a well-engineered structure for the sessions is a good combination for running focus groups. The biggest risk in using focus groups as a method to gather research data, is researcher bias (Kontio et al., 2004). This means that the moderator guiding the session needs to be completely objective and a good plan can help avoid this pitfall also.

3.5 Research Design and Implementation

The design of this research started three months before the data collection was done. In the beginning of 2015 four workshops were held over the period of two months in order to discuss about the scope of the research. The people participating in these workshops were the author, three researchers from the Department of Information Processing Science at University of Oulu and one representative from the case company. In these meetings the concept of lean, the current research and what is being done in the case company were discussed. It was told that lean is being studied and used in the case company on many levels from small improvement projects all the way to organizational transformation planning. Based on these discussions it was decided that the scope of this research would be on the level of studying how lean could be applied in software development, from the practitioners’ point of view. This was seen as a missing element that deserved further study.

To provide meaningful answers for the research questions presented in the previous chapter, the case study research method was selected. The case study method was seen as a suitable method for this research because it studies phenomena in their natural context and especially when the boundary between the phenomena and its context is not clear (Yin, 2009). In software engineering, the boundary between the studied object and its environment is not clear and understanding the interaction of these two can provide meaningful answers (Runeson & Höst, 2008).

The focus group method was selected as the primary means for conducting the data collection for the thesis. Archival data and observations in the case company were also chosen to support the data from the focus group sessions. The selection of focus group method for the data collection over one-to-one or group interviews was that because the concept of lean and especially lean software development is a bit fuzzy and no single correct answers exist. Because focus group allows the participants to build their thinking on top of the ideas of others and generate ideas in this way, it was a natural choice for data collection (Langford & McDonagh, 2003). The focus group method can
be used especially in software engineering to address the types of research questions that this research takes a look at, which is evaluating new concepts (Kontio et al., 2004).

The seven principles of lean software development by Poppendieck and Poppendieck (2003 & 2006) were selected as the theoretical framework for this thesis and the justification for the selection is presented under chapter 2. The principles have been presented at least in two different orders and the naming has been changed from *Lean Software Development: An Agile Toolkit* (2003) to *Implementing Lean Software Development: From Concept to Cash* (2006). However, the content of the principles hasn’t changed, only the naming. The naming of the principles in this thesis is based on Poppendieck and Poppendieck (2006).

The case company is a large-scale software intensive company. The company makes products that are hardware and software. This means that software development in the case company is to a large extent embedded. This means that software is developed close to the hardware and the development efforts need to constantly take into consideration the needs of the hardware. The special characteristics of large-scale embedded development have been taken into account when analyzing the research data and reaching the conclusions. The case company was selected because it is a large-scale software development company and the benefits of lean can be reaped especially in this kind of large enterprises.

*Planning the focus group sessions.* The design of the focus group sessions followed the guidelines provided by Kontio et al. (2004) and Langford and McDonagh (2003). It was decided that two focus group sessions would be held. If there would be need for more, it would be decided later. This was based on the guidelines by Kontio et al. (2004), that more than one focus group session is held. A pilot session was also held before the actual sessions to evaluate the design of the session and improvements were made to the session structure based on the knowledge from this pilot. The focus group sessions were scheduled to have about a week between them in order to be able to improve the structure of the session for the second one, after learning from the first one. This is one of the main reasons for having at least two focus group sessions that you can learn from the first one and modify the structure or the content of needed. The focus groups were designed to cover three main topics. In order, these topics were: 1) Discussion on how the participants understand Lean Software Development, 2) Discussion on challenges that the participants face in their current way of working and 3) Discussion on opportunities and barriers for applying lean to software development.

To support the sessions, two moderators’ guides were created based on the template by Langford and McDonagh (2003). These guides are presented in appendixes A and B. The moderators guides contained the topics in order with a timetable for the sessions to ensure that time was allocated appropriately for each topic. In a nutshell the guides had a row with a topic or an action, the description of what it contains, listing for possible aids to be used, duration in minutes and the exact time when that topic should start in order to stay in schedule. The author acted as the moderator in each of the sessions meaning that he guided the discussion and probed deeper if needed. The job of the moderator is also to answer any questions and to make the participants feel comfortable during the sessions. For the moderator to be well prepared, lessons learned were discussed prior to the sessions with persons that had experience from working as a moderator in a focus group session. This helped in avoiding the worst pitfalls and to prepare for potential disasters. Because acting as a moderator is a full-time job, two assistants were helping in both sessions. The assistants took notes, reminded if time was running out and helped with the practical arrangements. The notes and post-its collected...
by the assistants were used as research data for the analysis. Slideshow was used to visually guide the sessions.

The participants selected were people working in different software related projects. In the first group there were three software engineers taking part. In the second session there were two software engineers, one software specification specialist and one testing manager. Because the focus of this research was lean software development, we wanted to know what the people working on the ground level had to say about the topic. Because lean can be seen to spread through all functions of a company, there really is no rule in whom to ask opinions or ideas from. Langford and McDonagh (2003) suggest that having smaller groups (four to six participants) may be more suitable because some people might be intimidated by larger groups and also the fewer number of people can help in creating a more comfortable situation where participants can be more productive and make them invest more in the session. Experience of an R&D line manager from the case company was used to select participants who would be suitable for the sessions. This resulted in the selection of participants with an improvement mindset and a will to discuss new ideas. Thus it can be said that the participants recruited were representative, insightful and motivated (Kontio et al., 2004). Information about the participants along with some general information about the focus group sessions is presented in table 1. This table lists both of the focus groups separately.

Table 1. Information about the focus group sessions and the participants

<table>
<thead>
<tr>
<th></th>
<th>Focus Group 1</th>
<th>Focus Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>10.6.2015</td>
<td>15.5.2015</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>3 hours</td>
<td>3 hours</td>
</tr>
<tr>
<td><strong># of participants</strong></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Participants titles and experience in the field</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW engineer, 10 years experience</td>
<td></td>
<td>Test manager, N/A experience</td>
</tr>
<tr>
<td>Technical lead (SW engineer), 9 years experience</td>
<td></td>
<td>SW engineer, over 7 years experience</td>
</tr>
<tr>
<td>Scrum master (SW Engineer), over 10 years experience</td>
<td></td>
<td>SW engineer, 2 years experience</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specification specialist (SW Engineer), 18 years experience</td>
</tr>
<tr>
<td><strong>Transcript length in pages</strong></td>
<td>25 pages</td>
<td>20 pages</td>
</tr>
</tbody>
</table>

Conducting the focus group sessions. Two focus group sessions were organized in the case company in order to collect the research data. The length of both sessions was three hours each, including one fifteen minute coffee break. The sessions were held at the case company’s facilities. Four participants were selected to participate in each session. Unfortunately for the first session, one participant didn’t arrive, leaving the total number of individuals taking part in the sessions to seven. Participants were guided to first write some initial ideas on post-its for each topic in hand, before starting the discussion. Coffee break was held in the midway of the sessions. After the break the rest of the topics were discussed and the sessions ended with a summary where the main points
discussed were summarized for the participants. Participants were also asked for any feedback about the sessions.

The rough structure of the focus group sessions is presented in figure 5. The sessions started with an introduction and explanation of the goals and ground rules of the sessions. After this the discussions started on the aforementioned topics. The topics were explained and the moderator answered any questions raised by the participants. The first two topics as explained before were self-explanatory and didn’t need much explaining from the moderator. However the third topic of discussing the opportunities and barriers in applying lean to software development needed some initial explanation by the moderator and this was recognized already beforehand. Lean Software Development was presented through the theory by (Poppendieck & Poppendieck, 2006). The seven principles of lean software development were presented one by one so, that after presenting each principle the participants got to discuss the opportunities and barriers in that particular principle and build ideas on how to implement them. The original aim was to cover four of the seven principles in the first session and the last three in the second sessions. This decision was made because discussing this kind of principles can take a lot of time to really get deep in the discussion and generate ideas. In the first session it was possible to go through all the seven principles within the given time. Based on this evidence, it was decided that all the seven principles would be also discussed in the second focus group sessions. The order of in which the principles were discussed in the second session was based on prioritization. This prioritization was done so that the principles that generated the most discussion in the first session were to be discussed first and the ones with little interest in the discussion were to be discussed last. In the second session time ran out and two of the principles weren’t discussed. These principles were Optimize the Whole and Deliver Fast (Poppendieck & Poppendieck, 2006). Like mentioned earlier in this paragraph, these principles raised only little discussion in the first focus group session so the priority for them wasn’t high for the second focus group session. This still, directly affects the validity of the research results on these parts. Another thing to mention here is that after going through the results of the second focus group, it seemed that the results were confirming the ones from the first focus group. Thus, it was seen that there would be minimal benefit of arranging a third focus group so it was decided that the two focus groups were enough.
During the sessions, several tools were used to collect the data. The primary data collection tool in the sessions was audio recording, which was done by using an audio recorder. A mobile phone was also used parallel to the handy recorder to work as a backup in case the recorder would have failed for some reason. In addition to these, the post-it notes done by the participants were collected and grouped for each topic discussed. In both sessions, one assistant was also taking notes and writing up the main points raised in the sessions. Like mentioned in the previous paragraphs, the main topics were summarized in the end of the sessions to ensure that the participants had been interpreted correctly. In case of misunderstanding, fixes were made.

3.6 Data analysis

The audio recordings were transcribed by the author. All the research material including the transcripts, post-its and notes were centrally stored into a qualitative analysis software tool called NVivo. The transcripts were then coded in NVivo. Coding in this context means attaching labels and tags on quotes from the transcripts and organizing these into different structures. This way the research data from both sessions could be easily synthesized under the same labels and analyzed as a whole.

The top level structure was formed by the research questions and the collected data was first coded under those three categories. Because the guiding theory in this research is the one of Poppendieck and Poppendieck (2006) and the principles discussed in the sessions are presented by them, the coding process then progressed by organizing the data under the research question categories to the seven principles of lean software development. Several iterations were done to refine the data. The post-its and notes
collected from the focus group sessions were also coded in the process. Also the general discussion about the challenges in the current way of developing software was coded according to the themes that were raised up during the discussions, such as quality, tools, decision making and schedules.

When the research data was coded according to the structure described above, it was further analyzed. Some of the central themes raised up and were noted. Also connections between different codes were marked and the reasons behind these connections were recognized. Looking at these connections behind coded items revealed interesting findings, much richer than the coded items alone. Cause and effect relationships were identifiable and listed as findings. Discussions about lean software development and about the challenges in the current way of developing software supported each other by enriching the data through links and similarities. Looking for repeating patterns between the two focus groups was also done and used to support the analysis.

When the research data is coded in a systematic way like this, it ensures that the data can be traced back to its origin and the whole process is transparent. This contributes largely to the validity of this research (Runeson & Höst, 2008). The fact that the audio recording from the sessions were transcribed by the author means that it was possible to dive deep into the research data already in the beginning and attach initial codes to interesting pieces of research data.
4. Findings

The findings of this research are presented in this chapter. The research questions are answered separately from each other in their respective chapters. The research questions and their respective findings are presented in the same order as listed earlier in this thesis. First we discuss the findings in *the understanding of lean software development by the practitioners*. After this we take a look at the *opportunities in applying lean software development*. Finally we also take a look at some *barriers in applying lean software development*. The nature of the last two research questions is such that some intentional overlapping of the results can happen when the results are presented. Direct quotes from the interviews are used when this is seen necessary to support the validity and rigor of the analysis.

4.1 Practitioners’ Understanding of Lean Software Development

This research question aims to give an explanation of how the practitioners understand lean software development without any prior introduction to it. This is important to understand because lean is still such a fuzzy concept that assessing the understanding of people on the subject is crucial. With this knowledge it is possible to see the variation in the understanding and also to identify if there’s something that is commonly seen as a key component in lean software development. The literature on lean software development hasn’t presented any research on the practitioners’ understanding of lean software development.

The practitioners’ understanding of lean software development was assessed simply by asking them to explain their view of what lean software development might be and how they interpret it. When needed, the participants were asked probing questions to help clarify their answers. Because the understanding was explored in the beginning of each focus group sessions, it is possible to say that the participants acted as a “blank canvas” meaning that they hadn’t been given any prior introduction or explanations about lean software development principles by this research activity. Of the total number of seven participants, only one person mentioned that “Oh yes, there are those seven principles or something but I can’t really remember what those were”. This was the only time anyone recognized the existence of the seven principles of lean software development prior to the point in the focus groups when the principles were introduced and explained.

The table 2 maps the seven principles and shows if they were identifiable from the discussions. Of the seven principles, three didn’t map to any statements but four did. The three principles that were not identifiable from the discussions were *Create Knowledge, Deliver Fast* and *Defer Commitment*. The four principles that were identifiable from the discussions were *Respect People, Eliminate Waste, Build Quality In* and *Optimize the Whole*. For these four, one example quote from the participants is presented in table 2 for each of the principles. It is no surprise that most of the discussion was about *waste*. A quote from a participant in the beginning of one session was that “I also have the same idea that waste elimination is the first thing that pops into my head when discussing about lean” Waste is one of the most recognized concepts of lean and is associated with it. It was also joked that when waste elimination is or has been pursued, people are just “getting rid of everything that’s not necessary”. This was
seen as a bad thing at times because it is possible to do this waste elimination from a narrow point of view, which then results in necessary activities getting eliminated.

**Table 2.** Mapping of the Lean SW principles to the findings related to understanding of lean software development.

<table>
<thead>
<tr>
<th>Lean SW Development Principle (Poppendieck &amp; Poppendieck, 2006)</th>
<th>Correspondence in practitioner understanding from the focus groups</th>
<th>Example quote from the focus groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Knowledge</td>
<td>No.</td>
<td>-</td>
</tr>
<tr>
<td>Deliver Fast</td>
<td>No.</td>
<td>-</td>
</tr>
<tr>
<td>Respect People</td>
<td>Yes.</td>
<td>“Putting focus on individual developers and teams so that they can decide their own way of doing. This would result in less bureaucracy.”</td>
</tr>
<tr>
<td>Defer Commitment</td>
<td>No.</td>
<td>-</td>
</tr>
<tr>
<td>Eliminate Waste</td>
<td>Yes.</td>
<td>“Basically my understanding of lean is that you need to be able to identify those targets where waste occurs.”</td>
</tr>
<tr>
<td>Build Quality In</td>
<td>Yes.</td>
<td>“When you’re fixing bugs, it includes the creation of a dedicated test case. This simply means that same mistakes are not repeated.”</td>
</tr>
<tr>
<td>Optimize the Whole</td>
<td>Yes.</td>
<td>“Maybe it has something to do with the delivery chain also so that the integration of the software is quick. Currently we face challenges when the feedback from integration is too long.”</td>
</tr>
</tbody>
</table>

A lot of discussion about the understanding of lean software development concept was raised that didn’t map to any of the existing principles by Poppendieck and Poppendieck (2006). The general feeling was that lean is a trend concept that keeps getting mentioned and pushed all around but that no concrete possibilities have been presented on how to apply it. It is most likely that this feeling is because lean software development is mostly about the principles and having the principles as a guide to achieve the lean way of working for each company. Because lean then means different things for different organizations in practice, people tend to think that it is just a trend word with no content because concrete lean tools or methods do not widely exist for product development. A couple of comments were raised with the suspicion that is not lean supposed to be just for manufacturing, not for product development.

A repeated finding in the sessions was the discussion on development tools. When discussing about what lean software development is and how do the participants see it, strong emphasis was on the importance of good tools. A quote from the sessions states that “The tools should be chosen to support the work and not the other way around”.
The participants strongly saw that lean software development would be about choosing the right tools so that they would support the work as well as possible and this is where big benefits could be gained from.

Wang et al. (2012) recognized that the concepts of agile and lean are not easy to distinguish from each other and they even have some overlapping. This was also recognized in the focus groups: “Lean and agile are not completely overlapping so that lean would have a wider scope than agile but agile defines things that lean does not. For example lean does not instruct you that how should you be handling your requirements and in what kind of pieces should you be developing software”. The assumption was that there would be much more confusion from the participants about the relationship between agile and lean but in the end it seemed that when the participants got to discuss their views together, they pretty quickly reached a consensus about the difference. The quote “Lean is not a software development process like agile (scrum) is” sums the thinking up pretty well and demonstrates that the participants understood the difference without the moderator explaining it to them.

It seems that lean software development is understood by the practitioners in many different ways. Four of the seven lean software development principles could be mapped to the discussion in the focus groups about the understanding of lean software development. This means that the basic concepts were more or less understood even though the participants had no idea about the principles. The applicability of lean to software development was questioned because of its origin in manufacturing and there was also a general feeling that the term lean is just used as a trend word. Eliminate Waste principle had most references in the coded data because if people know lean from something, it is the elimination of waste. Compared to agile software development, it was seen that lean software development is somewhat overlapping but is meant for tackling different kinds of problems than agile so there would be minimal conflict between the two ways of thinking and improving.

4.2 Opportunities in Applying Lean Software Development

To study this research question, each of the seven principles of lean software development was presented and explained in the focus group sessions one by one. To provide meaningful and holistic explanation of the principles several sources on the seven principles of lean software development were studied (Jonsson et al., 2013; Poppendieck & Cusumano, 2012; Poppendieck & Poppendieck, 2003, 2006). After each principle was explained the participants were given time to discuss the opportunities and barriers for applying that specific principle to practice before proceeding to the next principle. In the resulting discussions the participants started generating ideas of possible ways to apply the principle in practice in relation with the opportunities and barriers for doing so. This chapter takes a look at the opportunities found. By opportunities we mean things that can be done in practice to embrace one or several of the seven principles of lean software development. Opportunities can be seen as things that are more easily achievable. The key opportunities found are listed and mapped to the seven principles in table 3. Next the key findings will be explained in more detail, mapped to the seven principles of lean software development.
Table 3. Mapping of the Lean SW principles to the findings related to opportunities in applying the principles in practice.

<table>
<thead>
<tr>
<th>Lean SW Development Principle (Poppendieck &amp; Poppendieck, 2006)</th>
<th>Examples of opportunities from the focus groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Knowledge</td>
<td>• Fast prototyping to explore solutions: pivot</td>
</tr>
<tr>
<td></td>
<td>• Trash the prototype and use the knowledge gathered</td>
</tr>
<tr>
<td></td>
<td>• Root cause analysis</td>
</tr>
<tr>
<td></td>
<td>• Informal code reviews</td>
</tr>
<tr>
<td>Deliver Fast</td>
<td>• “We’d like to deliver fast also as developers”</td>
</tr>
<tr>
<td></td>
<td>• Implementing the best in class tools to enable fast delivery</td>
</tr>
<tr>
<td>Respect People</td>
<td>• Developers to step more on the specification field</td>
</tr>
<tr>
<td></td>
<td>• Role of a “technical leader” is needed</td>
</tr>
<tr>
<td>Defer Commitment</td>
<td>• Prevent inclusion of hack solutions to code base to reduce unknown dependencies</td>
</tr>
<tr>
<td></td>
<td>• Leave specifications “open” for developers as good quality drafts</td>
</tr>
<tr>
<td>Eliminate Waste</td>
<td>• Task switching and Defects are seen as most severe wastes</td>
</tr>
<tr>
<td></td>
<td>• Employees are happier with less waste</td>
</tr>
<tr>
<td>Build Quality In</td>
<td>• Right tools would encourage people to produce quality</td>
</tr>
<tr>
<td></td>
<td>• Coding standards and style guides</td>
</tr>
<tr>
<td></td>
<td>• Faster feedback from continuous integration systems</td>
</tr>
<tr>
<td>Optimize the Whole</td>
<td>• “Process Profiling”: Focus on optimizing the parts of the process that consume the most time</td>
</tr>
<tr>
<td></td>
<td>• Usage of E2E tools for managing knowledge</td>
</tr>
</tbody>
</table>

The first thing that came up when discussing Create Knowledge principle was that the concept of fast prototyping is a really important one. It means developing a prototype of the solution with minimal resources, demonstrating certain functionality. It can also be called a proof-of-concept, even though a prototype and a proof-of-concept might mean slightly different things. Developing prototypes helps the development team to explore different solutions to a problem by demonstrating that certain things can be done while others cannot be done. It is possible to assess the feasibility of different solutions or just simply provide a proof-of-concept that a certain implementation is possible not only in theory. By doing this kind of work the developers are able to gather data and then make decisions based on that data. This data can then be later used for various projects to see if something has already been prototyped and if this knowledge could be used again. In lean product development, limit curves are used as a tool for distributing this data (Ward & Sobek II, 2014). According to the practitioners, the thing to remember here is that “The most important step in prototyping is the last one; that you trash the prototype once it is done.” The aim is to create beneficial data, based on which decisions for further development can be done. It was commented that too often it is seen that a prototype ends up being the final product and because of its nature; defects and quality problems start to occur in later phases of development. So the rule is: once a prototype is complete, collect the data, trash the prototype and use the collected data to develop the solution from scratch.

Root Cause Analysis was one term presented for the participants under the Create Knowledge principle. The conclusion was that the practitioners are doing root cause analysis for bugs that are reported but it is not being done for the process. Analyzing the root causes of the process was not a formal practice that was in place. Rather it was so,
that sometimes the developers had found out intentionally or unintentionally that what
the thing is that’s causing problems in their process. If the cause has been something
that they themselves are not able to fix, the request for improvement has been put
forward but usually it stops there and the process root causes do not get fixed.

Another idea generated in the sessions was related to taking the prototyping “fail fast”
concept to a lower level, to everyday problem solving. It could be so that especially
when a junior developer would start developing a solution for a problem, he / she would
then after a day or two show their progress to a senior developer. The idea of this would
be for the senior developer to do a quick review of the solution to see if it is going
anywhere near towards a feasible solution. If the direction would be good then the
senior developer would give the green light to go forward and if not, they would
together think about a new direction for the solution. The term coined for this was an
informal code review. The benefits from doing this are clear because it can save the
developer from wasting their time on a solution that’s doomed from the beginning. To
some extent, this is being done at least in one team but it is no known practice. One
participant commented that in their project, everybody is able to see pending code
reviews and comment to them. “People must’ve learned a lot just by informally
reviewing other code, giving comments and reading them”, it was said. The project
team in this case had been working with some new technology and informally
reviewing other commits was a way for the community to learn about the possibilities
and pitfalls of the technology.

The discussion on Deliver Fast principle had more weight on the existing barriers side,
more of that in the next chapter. The consensus amongst the developers was that it is not
only so that top management is requesting fast delivery and developers wouldn’t want
to deliver fast. The developers would like to deliver as fast as possible, also. This way
there would be less partially done work, which is actually one of the wastes, listed in
lean software development principles. The bottom line still is that it is not simply
enough to say that “We need to deliver fast” but that infrastructure, such as tools,
processes and skilled people are needed to make it work. The developers would enjoy
being able to work with atomic pieces of functionality instead of big lumps, because
doing changes to small pieces of functionality is naturally much easier than solving
dependency problems in big pieces of code.

Under the Respect People principle a lot of good discussion was raised about the
opportunities. An issue was raised, concerning the synchronization problems between
specification and development. It was seen that the specifications were made in
somewhat isolation and then later delivered to the developers, who then felt that the
specifications were not in a usable format always. One of the reasons for this was that
the developers, testers and specification people were working separately from each
other in isolation. “It would be good that developers would step more in to the
specification field” was one comment given in the sessions. This simply means more
cross-functionality to the teams so that specifications could be developed in parallel
with coding so that information sharing would be possible. Synchronizing the activities
between specification work and code development would lead to the possibility to ask
for opinions from one and other so that the work done would serve the recipient in the
best way possible. Because currently things are specified well before development on
those starts, it raises questions in the development but in that point of time the
specification work has already moved on and time is wasted when they need to start
digging for old documentation and decisions to support the developers. The concept of
feature teams were in place but it seems that there is no true cross-functionality
embedded in all of them. On the contrary one participant working in a project with a
fresh software stack commented that they’re doing specification work and code development in parallel and it is working pretty well. This was commented by another participant that it is true that in projects with new software stacks cross-functionality is more easily achieved when compared to the work in legacy projects where only a handful of people might know all the “tips and tricks” of the implementation.

When the importance of leadership was discussed, the importance of technical leaders was emphasized in both focus groups. When probed a bit deeper about the importance of a technical leader for a project, the answer was that “We’d be totally lost without our technical lead”. Usually the technical leader in a project is a seasoned expert, a senior expert that has years of experience on the subject, whether it is a new fresh software stack project or a legacy one. The importance of existing knowledge is highlighted in the latter one. For other developers, the technical leader is like a walking databank because they are able to answer a wide range of questions related to the development work on hand and in this way blockers can be quickly solved. These persons usually also work as gatekeepers by doing code reviews and making sure that only quality is going out from the development. It was noted that “Especially in our case, because we are working with a lot of proprietary technology and APIs, it is not possible to search Stack Overflow for solutions and this is one part where the importance of a technical lead steps in”.

It is clear that dependencies in the code base or in the architecture create problems. They can force people to commit to certain solutions because the dependency simply dictates that things have to be done in a certain way. This is not good when looking at the Defeer Commitment principle. To get rid of these dependencies, the quality of the code that gets integrated to the code base needs to stay excellent. It was said that when people have to provide fixes to problems with tight schedules, it often happens that the solutions are not thought through and under the pressure from management; the solutions are hacked to work. This can mean several things, for example that the solution is not respecting the architectural rules and by doing so, might create dependencies into the code base that are hidden. When these kinds of solutions get integrated, the dependencies are not known until they show themselves as bugs or other problems later on. The answer for this is to simply have a stricter gatekeeping process in code review, so that these so called hack solutions do not get integrated. Of course it is more easily said than done. Some discussion about the root causes for this was done and one of the views was that people should have quality thinking embedded to themselves. This means that the hack solutions wouldn’t happen in the first place. This means that people developing the code should be proud to have their name linked to a piece of code. This was commented so that a lot could be learned from the world of open source software development, especially the Linux kernel development community, where people can add “signed off” lines to patches. By doing so these people simply state that they were a part of the development of that specific patch in one way or another and kind of endorse the commit to be of good quality.

Deferring Commitment could also be embraced by leaving specifications to a state where they are not completely closed, so that there’s enough room for the developers to make the lowest level decisions that do not affect the outcome, only the technical solution. It is difficult to draw a concrete line to what exactly is this “enough room” and this could be tackled by having working cross-functional teams where people could agree on these matters. On the other hand the specification can’t be too open drafts because in this case the developers then do not know what the specification is requesting for them to develop. When the specification leaves a bit of room for the
developer to play with, it can be seen as respecting the developers’ input and giving them trust that they can do the job.

Eliminate Waste principle discussed the different wastes in software development and ways to tackle them. Of the seven wastes of lean software development, listed previously in this thesis, two of them were seen as the most severe ones by the participants. These wastes were: Task Switching and Defects. These two are interrelated wastes and eventually form a vicious circle. When defects are found in testing, they are reported back to the teams for fixing. The team that’s responsible of the component, in which the defect was found, is responsible for providing a fix for the defect. However when teams that are developing new functionality get these fix requests, the developers have to jump from the task of developing new functionality into the task of fixing some old functionality. This creates task switching. Especially when the flow of fix requests is constant, developers that are responsible for fixing different defects are constantly switching between providing a fix and trying to develop new functionality that was promised. The end result is that new functionality development takes a big hit and for example sprint schedules go off track. “Then with all this rush, really stupid and easily avoidable defects occur under the pressure to deliver on time”, one participants mentioned. As a general comment about eliminating waste, it was seen that it is not only the management that is concerned about waste occurring in development. The comment of one developer was that “It is actually so that the developers are happier when there’s less waste in the development, less unnecessary work”. So it can be seen that elimination of waste is also in the best interest of developers, not only managers, because less waste means that the work they do is more meaningful.

Build Quality In is the second to last of the seven principles of lean software development. Test automation in general was seen to be in a pretty good shape and that there’s governance in place from top down. It was said that test automation is really well in place across the organization. Again the issue of having the best in class tools was raised in the discussion under this principle. “Having the right best in class tools in place would enable people to produce quality”, one person commented. It was seen that by having the best in class tools, the tools themselves would prevent the developers from implementing bad quality. The developers said that they would need faster feedback from the integration systems that did their code integrate as assumed or was there problems. With faster feedback from the continuous integration (CI) system it is possible for the developer to constantly know what the situation is with the committed code without delays so they can proceed to the next thing faster. To tackle some of the quality problems, it was seen that proper coding standards would be needed. This would embed quality into the code in the sense of maintainability. “When developing code you should always be thinking of the next person, the next developer taking a look and trying to understand your code” was noted in the sessions. Good coding standards and style guides implemented across the organizations software R&D would help hugely in building the needed quality into the code base.

Optimize the Whole also raised some good discussion. When discussing this principle, the importance of it was straightaway understood by the participants. “We are doing agile but we are not agile” was a comment given reflecting on adopting agile methods such as Scrum into the development but the organization as a whole not changing accordingly. Laanti (2012) puts it well: “An enterprise with many fast Scrum teams is like a machine with its inner parts spinning rapidly, with transmission to the system level failing. In practice, the rest of the organization may be unable to react to the rapidly produced code and fail in integration and putting all the code that has been
produced into use. From the lean perspective, excessive code that cannot be integrated and used is waste”. Another thing that was raised in the discussions was the handling of information. Concerns were raised that to some extent, good tools are in place but people are not using them properly. This means that for example with JIRA, people are exporting excels from it at some point and then some people eventually re-enter the information to the system. The comment was that this is done only because some people are more comfortable with using excels instead of the dedicated tool. With strict guidelines this could be avoided and knowledge could be managed in an end-to-end (E2E) manner throughout R&D.

Process profiling was also one concept coined in the sessions when discussing the Optimize the Whole principle. The name comes from the activity of code profiling, which means that one profiles the code to find the functions that have the largest runtime in the system and focuses on optimizing those functions. Process profiling similarly means that by recognizing the parts of the working process that take up most of the developers’ time, the optimization efforts could be prioritized. This of course can be scaled up from the single developers work activities to organizational processes. In this way the biggest benefits from optimization and improvement activities could be gained.

This chapter presented the findings from the focus group sessions, mapped to the seven principles of lean software development (Poppendieck & Poppendieck, 2003; Poppendieck & Poppendieck, 2006). Possible opportunities of different thinking and concepts were presented and will be later concluded in chapter 5 together with the findings from other research questions. It can be seen that there were opportunities identified for all of the principles, more for some and less for the others. Next chapter works as a counterpart and a natural continuum by presenting the barriers in embracing these principles.

4.3 Barriers in Applying Lean Software Development

This chapter provides the answers to the third research question: “What are the barriers of applying lean software development?”. By barriers we mean things that create a barrier for embracing one or several of the seven principles of lean software development. Barriers can also be seen as things that are not easily achievable and thus present themselves as barriers. The key barriers found are listed and mapped to the seven principles in table 4. Next the key findings will be explained in more detail, mapped to the seven principles of lean software development.

Barriers for Create Knowledge principle are many. The possibility of exploring different solution alternatives through fast prototyping in order to gather data based on which to make further decisions about what to develop and how, was seen as a necessary thing amongst the participants. The barrier for this to work is simply the lack of time and resources. It was agreed amongst all the participants that extremely tight deadlines for delivery do not allow any experimentation with alternative solutions in order to find the best one. The result from this is that usually the solution that first comes up gets implemented without too much data to support the decisions to go with that specific concept. Taking this a bit further, the participants saw that doing these quick go-decisions in development without proper knowledge leads to quality problems because of bad architecture or implementation.
Table 4. Mapping of the Lean SW principles to the findings related to barriers in applying the principles in practice.

<table>
<thead>
<tr>
<th>Lean SW Development Principle (Poppendieck &amp; Poppendieck, 2006)</th>
<th>Examples of barriers from the focus groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Knowledge</td>
<td>• Coding standards missing</td>
</tr>
<tr>
<td></td>
<td>• No time to explore solution alternatives</td>
</tr>
<tr>
<td></td>
<td>• Process root causes not fixed</td>
</tr>
<tr>
<td>Deliver Fast</td>
<td>• Missing code base quality</td>
</tr>
<tr>
<td></td>
<td>• Big chunks of functionality</td>
</tr>
<tr>
<td></td>
<td>• Do the internal and external customers want fast delivery</td>
</tr>
<tr>
<td>Respect People</td>
<td>• Decisions made on upper level and pushed down</td>
</tr>
<tr>
<td></td>
<td>• Promises are made by people with insufficient knowledge about development</td>
</tr>
<tr>
<td></td>
<td>• Transparent process for teams to move their process improvement suggestions non-existing</td>
</tr>
<tr>
<td></td>
<td>• Globally distributed teams</td>
</tr>
<tr>
<td></td>
<td>• Decision making “narrow sighted” on lower level</td>
</tr>
<tr>
<td>Defer Commitment</td>
<td>• Price usually grows as commitment is postponed</td>
</tr>
<tr>
<td></td>
<td>• Legacy and architectural dependency problems</td>
</tr>
<tr>
<td>Eliminate Waste</td>
<td>• Might lead to elimination of necessary activities</td>
</tr>
<tr>
<td>Build Quality In</td>
<td>• Embedded development is challenge for TDD</td>
</tr>
<tr>
<td></td>
<td>• Lack of Quality thinking</td>
</tr>
<tr>
<td>Optimize the Whole</td>
<td>• Synchronizing work between teams</td>
</tr>
<tr>
<td></td>
<td>• Sub-optimization: Having agile teams in a non-agile company does not reap the benefits gained on the lower level</td>
</tr>
<tr>
<td></td>
<td>• E2E understanding of the value stream missing</td>
</tr>
</tbody>
</table>

Work standardization is another topic under the Create Knowledge principle. Related to this the concern was raised about missing coding standards and style guides. Because a unified way of coding is non-existing, teams and projects end up creating their own standards, which then create a huge amount of confusion when working in new relationships between each other. Another thing related to standardized work was the discovery that while in some teams it is simply impossible to get your commit integrated into the code base without code reviews, in some teams this is possible and is not monitored. It does not take much to realize that this undisciplined way of gate keeping results as for example quality problems later on.

Root cause analysis is yet another topic under the very same principle discussed earlier. When root cause analysis is done for bugs found from the code, the aim is literally to analyze the root cause for the bugs so that those bugs would not be repeated in the future and at the same time reveal possible hiding quality problems. A participant commented that “Yes we are doing root cause analysis for our bugs and we have pretty good coverage also. The only downside is that the quality of this analysis is not very high”. Another participant replied that: “I think the reason why the quality has gone down is that the root cause analysis has started to feel like a forced process that we have to do”. It is easy to see that when the developers start to feel that there’s no longer any benefit of doing something but rather it is a mandatory part of their work that has to be done to get it out of the way, people start to lose interest in doing that analysis. Related to the same conversation it later revealed that it is not always even the developers that are the owners of the errors who do the root cause analysis but it can be done at a different site in another country. This was simply then seen as waste because the developers do not learn anything if someone else is doing root cause analysis for their
bugs. To get the root cause analysis practice to work, the developers need to start feeling that doing it is actually beneficial and this could be possibly achieved by prioritizing the things to analyze and leaving some minor faults unanalyzed.

To enable *Fast Delivery* principle, a few barriers need to be taken down. The thinking was that the number one reason why fast delivery is not possible is that the code is developed in big chunks instead of small functional pieces. It is pretty simple that the smaller pieces you are developing, the faster you are able to get one piece completed and as a results, steady flow on functionality goes out. The root cause for not being able to do this was seen to be in the tools in use. It was said that for example the version control system that was used didn’t allow the development and committing of small pieces of software for various reasons not explained explicitly. Comments were that simply by deploying a better version control system, it would be possible to start moving towards smaller functional pieces of software. Although, it was also mentioned that sometimes the architecture might also lack modularity and this can be the reason for big chunks of code.

The nature of the industry the case company is doing business in, was also discussed as a factor affecting the possibilities for *Fast Delivery*. The developers mentioned that the delivery of embedded software has always been such that it happens a few times a year and they were a bit suspicious that would the customers really want new functionality delivered all the time. Also, inside the company R&D this kind of customer relationships exists when some team is serving another team and so on. The participants wondered that is there then any benefit of delivering fast if the next party cannot make use of the delivered item until later on in time. For this to work, the entire R&D has to be very well synchronized so that when delivering fast, everyone is able to react to it and make use of the delivered items.

The principle of *Respecting People* raised a lot of good discussion, especially on the barriers for embracing it. The general feeling was that a lot of times decisions get made upstream and are pushed down. This happens even when the teams thought they had the responsibility of deciding on the matter and this creates confusion. Related to this problem, it was seen that usually these decisions are made by people that have insufficient knowledge on the topic and should not be having the right to make those decisions. Related to this matter, the reason for too tight, almost impossible deadlines was seen to be that people with insufficient knowledge about development work are making promises for delivery schedules. Development teams should be listened more when making these schedules because at least they are able to give pretty good estimates about the time needed for developing a certain piece of software. This does happen time to time according to the participants but too often the effort estimates are translated into deadlines, which is exactly what’s not supposed to happen. The result is that of course the teams deliver the requested work on time but because of insufficient time, the deliverables can be of bad quality since shortcuts have had to be made. On the contrary it was also mentioned that there exists a risk in pushing too much of the decision making to the lowest level because “the decision making might be too narrow sighted and some obvious things might be left unaccounted for”. This is of course true and a balance need to be found with who is responsible of deciding about what and when. Globally distributed teams was mentioned to be one big blocker for moving the decisions making to the lowest level because it is hard to ensure that people all over the globe have the sufficient knowledge to make those decisions.

To truly *Respect People* to work, the people need to have a clear way of getting them heard. The feeling was that people do not really know what the correct way is if you
want to suggest improvements. For example Scrum retrospectives are done but the improvement ideas that require external actions get stopped to the first manager and things do not improve. The reasons for this can be many, for example that the managers simply do not themselves have the power or the vehicle to facilitate those improvement ideas. People get frustrated when they are left feeling completely powerless with their improvement ideas not moving anywhere. Having a transparent process for the improvement ideas would enable the developers to see how the idea is progressing and who is responsible of acting on it.

*Defer commitment.* In software development it is possible to defer commitment up to some extent and it can be beneficial. Still, the participants emphasized the fact that this has to have a limit. One comment was that “For example when developing software with an object-oriented language, it is possible to create really well adapting interfaces into the code but the more adaptability you want to embed into it, the more it costs”. This means that the creation of super adapting components is not always, if ever, ideal because the price tag on those can be really high. Also as mentioned before in the opportunities section, having those dependency problems in the code has a huge effect on how much it is possible to postpone commitment. With a fresh software stack, commitment is possible to be postponed a lot but when you are dealing with legacy implementations, the architectural and hidden dependencies create problems and you might be really committed to certain solutions right from the start unless a massive cleanup and refactoring effort is done to the code base.

*Eliminate Waste* didn’t raise too much discussion about the barriers for it. The biggest concern that kept coming up is that if individual waste elimination activities would be initiated, how it would be guaranteed that no necessary activities would get eliminated in the process. The reason for this concern was the fact that different stakeholders might see some activities as wastes while some sees them as necessary. This means that the question about the perception of waste is present. Conclusion still was that people think eliminating waste is a good idea, as long as the different views from relevant stakeholders are taken into account when pursuing these activities. Important to remember with this principle, is that waste elimination should not be done as an isolated activity but the waste should be eliminated as a result of improvements in the working process. When waste elimination is seen in this way, it poses much less threats that would concern the elimination of necessary activities.

The principle of *Build Quality In* faces also barriers. One of the most discussed facts was the challenges with the test driven development (TDD) approach. TDD means that the tests are written first and then, and only after then the actual code is developed against the test until the test passes. With the embedded software development context, there is a problem however. The participants put heavy emphasis on the fact that sometimes such mockups are needed that their complexity is greater than the functionality of the actual code. Another view on this was that “For example an active algorithm waits for parameters from the hardware which it then uses to configure something. In this kind of cases it is hard, if not impossible to realistically simulate the hardware with a test”. Partly because of these things, the result is that code is developed and unit tests for the code are written afterwards which is not following the TDD practice at all, it is the opposite.

Final principle and its barriers is *Optimize the Whole*. According to the participants often times functionality is developed with a rush, only to realize that it ends up waiting in queues. The reasons for this are many, such as problems occurring in the integration activities or need to synchronize with other parts of development. The problems with the synchronization should be easily avoidable according to the participants but would
need changes. These changes include things like defining all the interfaces so that there is no need to synchronize with other teams. This again is of course more easily said than done. Completing items and not putting them into use is not following the just in time development (JIT) thinking (Womack & Jones, 2013). In general there was also some discussion about the threats of sub-optimization on all levels. Some comments were raised about the fact that improvement is done on various activities but what is the point of these optimization efforts, if the next step is then stopping the flow. To get the real benefits from improvement activities, one has to be looking everything from and E2E perspective so that the value would flow all the time, across all departments, not just single ones.

This chapter presented barriers that were identified in the case company for the seven principles of lean software development. Barriers were identified for all of the principles. Next, the results of this study are discussed with the existing research in the field.

4.4 New Findings

This chapter presents findings from the focus groups that were not identified from the research by Rodríguez et al. (2013) and Poppendieck and Poppendieck (2003, 2006). The biggest reason for these new findings is most likely the fact that the scope of the focus groups was on people from the lowest level. The participants were mostly software engineers, one of them being a test manager. For this reason the discussion and the ideas generated from this discussion concern practical things. The scope of previous research on the topic of lean software development has mainly focused on interviewing people from higher levels of organizations and this is of course reflected in those results. For example when Rodríguez et al. (2013) studied lean software development at Ericsson, developers were only one part of the participants. Other groups were fault handling team, tools team, management team, agile coaches team and product owner team (Rodríguez et al, 2013). For this reason the scope has been much wider than purely investigating the developers’ views.

Not all of the findings related to opportunities were found from existing literature meaning that these findings are new. One of the ideas generated in the focus groups was harnessing informal code reviews to help in making sure that people are developing their solutions according to feasible designs. What this would mean in practice is that especially when a less experienced developer starts developing a solution and doing design for it, a senior person in the team such as a technical lead would then do an informal inspection on the work after some time. The senior person would use his or her experience to evaluate the feasibility of a potential design to identify any problems with it already in the early phase. It was discussed that this wouldn't require too much time but would yield massive benefits when non-feasible solutions wouldn’t get done. In this way junior developers would receive continuous feedback and the gatekeepers that do the formal code reviews could trust that at least the feasibility of the solutions is checked in early stage. This approach allows the possibility to evaluate solution alternatives and make knowledge-based decisions. Additionally to support this same thinking, keeping the code reviews open for every developer working on the same project could be a beneficial way to amplify learning, especially when using new technologies such as new programming languages.

Another opportunity that was recognized in the focus groups was the possibility to do “process profiling”. This term comes from the term code profiling which mean identifying the most time consuming parts of code and optimizing those first. With the
same thinking, *process profiling* would mean prioritizing the improvements on the parts of the process that constitute of the most time. In this way, sub-optimization could be partly avoided and focus would be there, where the biggest gains can be achieved from optimization and improvement efforts. The concept is somewhat similar to value stream mapping but the basic idea is a bit different because *process profiling* would be only used to prioritize improvement points.

Some barriers were found in this study that could not be identified from the existing research. One category of barriers that was found as a challenge in this case was the quality of the code base. This discussion was related to the fact that defects and dependencies often hide in the code base because of bad quality. This was commented to be because of people lacking “Quality thinking” attitude and that building a culture of quality thinking would be needed for the software quality to grow from its current state. The thinking was that because of too much pressure and too tight deadlines, people end up creating hacked solutions, which are not necessarily respecting the architectural rules. These bad solutions then get integrated into the code base because of insufficient gate-keeping, such as in-depth code-reviews. This creates a low quality code base, which causes problems that stack up as the time goes by. As the technical debt there grows, the more expensive it becomes to fix it. And by fixing here we mean refactoring, or worse; remaking the code. This is no easy task and all actions should be taken to prevent this as poor quality code base creates barriers for improvements. If the quality of a code base can be maintained on a high level, this enables better predictability into the development and less firefighting of unexpected problems.

The power of focus group sessions was also identified as a mentionable fact. Much of the previous research has had its focus on an upper level as mentioned before. In addition, some direct lean software development methods have been introduced such as *Kanban* and *stop-the-line* principle. These have been explained previously. What is meant by the power of the focus group method here in this case is that it seems that following the seven principles of lean software development and arranging developers to discuss about those topics in a focus group session could be used as an improvement method in companies. The results of this study point out that one of the ways for organizations to apply lean software development, would be to organize these kinds of sessions. It has been said that each company has to make its own journey to lean (Ward & Sobek II, 2014; Oosterwal, 2010). This kind of an approach could be used to drive this journey by bringing the improvement opportunities and barriers from developers upstream to management. It would require further study to find out how the suggested approach would benefit strategic transformation planning.

### 4.5 Feedback from the Case Company

The findings from the focus group sessions were presented in the case company for an improvement initiative. This received feedback contributes to the external validity of the research as it shows that the findings of the study are useful and partly confirm the results of previous studies done in the company. Three sessions lasting from one to two hours were conducted where the stakeholders participating were many. People from R&D management and management of improvement initiatives participated in these sessions. The findings were presented in the sessions and then discussed by reflecting to the previous knowledge on the subject. Future directions were eventually agreed to possibly use some of the findings to improve current practices.

The findings presented in these sessions were related to concrete things found to be wrong in the current way of developing software. These findings weren’t framed with
the lean software development principles but were more concrete improvement items. In these sessions it was identified that some of the findings were confirming evidence for already done studies on software development challenges. Some of the findings, however, were completely new. It was decided that these new findings would be taken under the microscope for further study in upcoming improvement initiatives.

The feedback from the stakeholders involved in these aforementioned discussions was really positive. It was commented that the confirming evidence along with the new findings provided useful information for planning future improvements. In addition, using the focus group method as an alternative for one-on-one or group interviews was something of interest. Especially the fact that the participants are free to discuss around a specific topic and generate ideas was seen to be a beneficial way of gathering information for improvements.
5. Discussion

In this chapter the research results will be discussed with existing research. The chapter will discuss the research questions in the same order as presented previously, starting with the understanding of lean software development by practitioners. This chapter will present similarities and differences between the existing research and the results of this study.

In the focus group sessions, the participants discussed about how they understand lean software development. Of the seven lean software development principles, four could be mapped to the discussion presented by the participants (Poppendieck & Poppendieck, 2003, 2006). These were Respect People, Eliminate Waste, Build Quality In and Optimize the Whole. Mapping to the remaining three principles (Create Knowledge, Deliver Fast and Defer Commitment) wasn’t possible because no discussion was raised related to those. When Rodríguez et al. (2013) studied lean thinking in a large-scale software intensive company they identified four main elements for lean software development. These were Value and Value Stream, Flow and Pull, Perfection and People. The results of our study show also that value and value stream were identified by the participants as the core idea in lean. Also the People factor was seen as fundamental for lean. Flow was also identified up to some degree but Pull and Perfection were not explicitly mentioned by the participants to be of importance. Notable here is that our study is discussing about how the practitioners understand lean software development without any introductions and the research by Rodríguez et al. (2013) presents the main elements that were seen as the most important ones from the existing sets of principles of lean. Hence, any generalizations on this topic are not possible. Assessing the understanding of lean software development hasn’t been researched much so the discussion with the existing research is somewhat limited. However this study suggests that practitioners see creating value as the core element of lean. Related to this, eliminating waste was also known as an important concept. Also the importance of a working delivery chain was mentioned and seen as a part of lean software development in ways such as working continuous integration and test automation. Then the importance of giving power to the people was emphasized as an integral part in lean. The importance of good tools was mentioned by the participants as something that lean software development might be. This can be actually mapped to the three categories of lean product development by J. Morgan and Liker (2006) who identify People, Processes and Tools as the main categories to focus on. All of the discussion about the understanding on lean software development from the focus groups can be linked to these categories, partly because the categories are really general and can cover a whole lot of things. The discussion about agile and lean showed that it was understood that there’s a difference between these two concepts. “Leagility” has been studied by Wang et al. (2012) and they’ve come up with six categories of lean application in agile software development. Based on the discussion from the focus groups in our study, it can be seen that the participants saw application of lean as Lean within agile so that various lean elements can be used to improve agile processes. This is one of the six categories recognized by Wang et al. (2012).

The second research question took a look at the opportunities in applying lean software development. This was done through discussion based on the seven lean software development principles (Poppendieck & Poppendieck, 2003, 2006). Several
opportunities for applying the principles were found in the case company. The main findings are listed in the table 3. Rodríguez et al. (2013) studied the elements of lean that are more easily achievable in software development context. This is roughly the same as identifying opportunities in applying the principles to practice. Creating a culture of continuous improvement, Involving people in the transformation and Creating a team culture were identified by Rodríguez et al. (2013) as the elements more easily achievable. In our study the participants felt that especially involving people from the lower level to the transformation and improvement activities more heavily would yield great benefits. So, a similarity in the thinking can be seen here. The thinking was also that people are motivated to make improvements because when something works better, not only the managers but the developers too are more satisfied. This can be identified from the research by Rodríguez et al. (2013) also as the creation of continuous improvement culture. Basically people are ready to make improvements continuously because they enjoy working more when things are going smooth and that works as enough motivation for it. The results of our study also show that people see waste elimination as a positive result of improvement activities because the less waste there is, the happier the employees are. This same thinking applies also to the results about fast delivery so that delivering fast does not only make the managers happy but the developers would also enjoy more if they would be able to deliver faster. As Rodríguez et al. (2013) also identified, there are challenges in what improvements get done. As they state that many improvements are done based on the fact that they focus on daily issues inside the team. Many of the improvements that do not get done are issues that reach outside of the team. This was the case also in our results that the developers felt powerless when it came to improvements needed that reached outside their team. The general feeling was that these improvements get stopped to the first manager. The developers in our case also felt that the things are pretty well setup in their teams but the improvements that are needed are almost solely reaching out of the teams and requiring someone else to act on them.

The third and final research question was then about the barriers for applying lean to software development. Rodríguez et al. (2013) studied the elements of lean thinking that are challenging the implementation of lean software development. Again it is not exactly the same question as ours, but it is really close so the possibility to reflect these results exists. Rodríguez et al. (2013) identified three categories of challenges: Achieving flow, Transparency and Create a culture of continuous learning. Continuing from the ideas of the last paragraph, the studies agree that there is clear need for more organizational transparency to facilitate improvements. In their studies, Rodríguez et al. (2013) recognized that achieving flow in an E2E manner in product development is difficult because of the number of decisions needed there. Additionally the results were that all of the company needs to work in the same lean mode to really reap the benefits. Also large work items and work in progress (WIP) were found as blockers for achieving flow. The results of our study also confirm all of these things. Additionally it was seen that too many decisions are made too early so that the development is later on stuck with irreversible wrong decisions. Our study also raises the fact that sub-optimizing can be harmful. For example only implementing scrum teams with the rest of the organization not being able to react to their output makes no sense. This is what the study by Rodríguez et al. (2013) refers to by saying that “whole company needs to work in the same lean mode”. This is also referred to by Jonsson (2012) that “We should not just try to optimize a part of the process, for instance software testing, but instead look how the parts can be made fit together”. Jonsson (2012) then continues discussing about the relevance of systems thinking in lean software development because it can help explain “how things influence one another with a whole”. The results of our study show
then that learning through experimentation and evaluating solution alternatives is possible to some extent and gets done, but usually pressure from delivery schedules and management do not allow any extra experimentation in development. Rodriguez et al. (2013) identified the same problems. Because of the lack of time and pressure there was rarely time for conducting simple experiments to learn about the solution. To be able to create a learning organization, people must have the necessary time to be able to conduct experiments in R&D work because learning takes time and does not happen in the blink of an eye. The challenges with transparency and continuous learning have also been identified by Suomalainen et al. (2015) when they studied the challenges faced by ICT companies in their lean transformation initiatives. In addition they identified Strategy, Customer value, Organizational structure, Organizational culture and leadership as issues in transformation. In our study these weren’t studied in too much depth, mainly because the scope of this study compared to the one by Suomalainen et al. (2015) is much narrower in a sense, with the focus mainly on R&D team level issues compared to organizational transformation.

This study has confirmed many results from the existing research. It has also provided totally new knowledge about the practitioners’ lean understanding that others in the field haven’t yet studied. Because the focus groups of this research mostly consisted of developers, it can be seen from the results that the interest is in the details and in the day-to-day activities taking place in R&D. On the other hand, the results prove that developers are able to have quite analytic discussion about the things that are needed for organization-wide improvements to happen and why those things are needed. Next chapter presents the conclusions of this thesis.
6. Conclusions

This thesis studied lean software development in the context of a large-scale software intensive company. Three research questions were answered. These research questions were: “What is the practitioner’s understanding of lean software development?”, “What are the opportunities for applying lean software development?” and “What are the barriers for applying lean software development?”

A literature review was conducted first to determine the correct research questions mentioned above. The literature review presented relevant literature to lean software development. Literature from the history of lean in manufacturing, lean product development, lean software development, systems thinking, enterprise transformation and the relationship of lean and agile was presented. This literature review provides a holistic view on lean thinking and offers the possibility to really understand the depth that lean can have in organizations. Understanding of enterprise transformation, for example, is needed to understand the position of lean software development in the organizational context. The literature also pointed out the need for further research around the research questions.

Based on the literature review and the research questions derived from it, it was decided to conduct a case study in a large-scale software intensive company. The research data was mainly collected by organizing two focus group sessions. Observations and archival data were used as supporting evidence. The discussion in the focus groups was focused to provide meaningful research data to be used to answer the research questions. The framework for the discussion was the seven principles of lean software development (Poppendieck & Poppendieck, 2003, 2006). In total seven subject specialists participated the focus group sessions. The sessions were audio recorded and assistants took notes.

The audio recordings were transcribed and organized together with the notes and other material from the sessions. Qualitative analysis was done by coding the research data with the NVivo tool. The parent coding followed the structure of the seven principles of lean software development. Other codes, such as child codes were assigned when seen necessary.

The results provided interesting answers to the research questions. Four of the seven principles of lean software development were identifiable from the discussion about the understanding of lean software development. The importance of focusing on value and waste elimination were emphasized by the participants. As something from outside of the principles, the importance of selecting best in class tools for the development work to be used across the software functions was seen as an integral part of lean software development.

Opportunities for applying lean to software development were also discovered. Having a root cause analysis process for the way of working would be really beneficial. This would help in having transparency in moving improvement ideas from the team level to upper levels of management. To implement this kind a process would require of course commitment from several stakeholders that the improvement ideas will be actually
handled and taken care of. Another finding was that there would be great value in doing fast prototyping to explore solution alternatives. With this it would be possible to make decisions about development based on actual knowledge. This could result in increased quality of the delivered solution because it would remove some of the commitment that gets taken currently to specific solutions. The experimental approach would also free the developer from the chains of too strict, ready-made specifications. The general feeling was also that the developers were ready to be part of the possible transformation activities because anything that eases their work, they’re willing to participate. This is an opportunity that should be recognized and taken advantage of by engaging people from the team level to improvement activities, for example by interviewing them about the pros and cons of their way of working as was also done in this research.

Finally, the barriers of applying lean to software development were studied. Some of the most notable barriers are mentioned here. Sub-optimization was recognized as a barrier for achieving lean software development. The concern was that agile teams are working in an organization that’s not able to react to them accordingly. This means that work gets done towards a deadline but when it is completed it ends up standing in a queue and from the lean point of view this is waste. Hence, achieving flow can be seen as a barrier. To achieve the best gains in efficiency, software development should be optimized from end to end, avoiding sub-optimizations and bottlenecks that eliminate gains achieved on a preceding level from optimization of the process. Discussion about impossible deadlines being laid out in R&D pointed out that bad quality is usually a result of this. When the people do not have the sufficient time learn about the solutions but instead are under pressure to deliver all the time, bad solutions get implemented and bad quality gets integrated into, for example the code base. As solutions need to be mashed together with hurry, the amount of technical debt also grows constantly, which at some point needs to be paid out; one way or another. This is an obvious barrier for lean software development. Decision making was another of the major topics discussed. The consensus was that decisions get made by people with insufficient knowledge in wrong places. The end result is that developers have to adapt to decisions made on upper level which restricts their freedom in R&D. Especially if decisions are ready-made with insufficient knowledge, this causes problems and even bad quality if those decisions can’t be undone. The fact that developers do not always have enough decision-making power is a barrier for lean software development.

It seems that lean software development is much more than just methods or tools. Like agile software development, there are some tools and methods existing, but these are based on the underlying philosophical thinking and principles. There are many more off-the-shelf tools and methods coming from agile than there are from lean. Lean is much more than ready-made methods or tools. Lean is a way of thinking and guiding the improvement work with existing principles. By following these principles and thinking, it is possible for companies to find their own way of being lean. The small number of existing lean software development tools and methods is because of the fact that each company is different and has to take their own learning journey to lean. By deploying and following existing tools or methods without learning, it is very likely that one ends up with much complexity and little desired benefits. Study the principles and learn how you could implement those principles in your company. This thinking that each company has to take their own journey to lean has been identified by Oosterwal (2010) and Ward and Sobek II (2014). Poppendieck and Poppendieck (2003, 2006) also provide the same kind of interpretation that lean thinking can be used as guiding principles to facilitate improvement efforts. Wang et al. (2012) identified the usage of
lean principles to guide the adoption of agile practices as one of the main categories of combining lean and agile in software development domain.

Next the limitations of this study are concluded along with the directions for future research, in their respective sub-chapters.

6.1 Limitations and Threats to Validity

This chapter will list out the limitations of the research, which the reader needs to acknowledge when interpreting the results and conclusions. Concerning the external validity of the research, it has to be understood that lean software development was studied in the context of one case company. This poses a limitation to the possibility to generalize the results outside the case company to other companies. Rodriguez et al. (2014) mention that: “there is no universally accepted definition of Lean in software development”, so it can be seen that the study brings new insights to lean software development anyway. Analytical generalization is possible by extending the results to cases with similar characteristics.

The second limitation comes from the possibility to interpret the seven lean software principles in different ways by the researcher and the participants. This poses a threat for the construct validity (Runeson & Höst, 2008). To tackle this threat, the principles were explained according to the literature (Poppendieck & Cusumano, 2012; Poppendieck & Poppendieck, 2003, 2006). Also the concepts behind each of these principles are in the end of the day of a quite general nature and can be easily understood by participants in a limited time. In the end of each focus group session, the notes taken by one of the session assistants were presented to the participants in order to correct any misinterpretations.

Using the focus group method for data collection creates a threat for the collection of meaningful research data. This concern arises from the fact that the aim of the focus group session is to give the power to the participants by not asking direct questions and collecting answers but by letting the participants to discuss and build on top of the ideas of others. This means that without proper guidance from the moderator side, the participants might end up discussing unrelated topics, which would eventually lead to the collection of irrelevant research data. This concern was tackled by creating a proper moderators guide to be used during the session. This guide helped the moderator to keep on track. Also the assistants in the sessions helped to take care of the timetable. In addition the moderator was a socially strong person so he was able to quickly react and guide the discussion back on track in case it started wandering. Kontio et al. (2014) emphasize the importance of a strong moderator in focus group sessions.

In focus group sessions it is possible that participants can have hidden agendas, group dynamics might affect the discussion, social acceptability might be a problem, secrecy issues might exist or comprehension of the topic might be limited due to its complexity (Kontio et al., 2004). All of these factors can have a negative effect on the results. In the sessions, the moderator had a good control over the flow of the sessions. The only downside being that discussion on two principles in the second session was missed due to lack of time as mentioned earlier. The people participating in the sessions all got roughly equal amounts of chances to speak. Because of the nature of the topic which is related to process improvement and improving ways of working, there should not be any concern with participants having hidden agendas since all participants worked in the same case company. Because the principles discussed are of quite general nature,
limited comprehension because of the complexity of the topic was not an issue. There are no right and wrong answers.

During the research process, process documentation was studied in addition to the focus groups. Adding to this, the researcher participated in several meetings in the case company related to improvement initiatives and observations from these were used as confirming evidence. Some of the findings from the focus group sessions were discussed at the case company in improvement meetings and were found as either confirming previous findings done inside the company or as completely new ones that would be considered to be accepted as improvement items. This contributes to the external validity of the research. Because the findings of this study were seen as useful and planning to take them into use is ongoing, it can be said that this provides a big contribution to the validity and credibility of the research. The research data can be traced all the way from data sources, to coded data and to the results and conclusions. This process has been transparent and so it contributes to the reliability of the research.

By conducting a case study in a large-scale software intensive company on lean software development and collecting data from focus groups by having the seven lean software development principles as the backbone theory, should enable the possibility to reach similar results as in this study.

6.2 Future Research

The findings of this research provide various possibilities for future research on the topic. This research provided knowledge about how practitioners initially understand lean software development and what are the opportunities and barriers that exist in embracing each of the seven principles of lean software development in the case company. These things are important to be identified but instead of looking at lean software development with such a wide scope, future research could focus on exploring some of the findings in this research that offer the possibility for a narrower research scope.

For example studying the concept analyzing root causes in the process could offer various research possibilities such as implementing necessary processes and infrastructure to create the pipeline needed for identifying and fixing root causes in the process in large-scale organizations. Another topic of interest could be finding out what is needed to deploy a development environment with the possibility to explore with various solutions alternatives and then collect and reuse the data gained from this experimentation. This kind of research would have to battle with things such as managing the delivery deadlines of the project while maintaining the data creating experimentation up and running. More concretely, it could be asked that what the limit curves from lean product development are in lean software development context; how could those be implemented and utilized in a beneficial way.

One more topic of interest from the results is studying what kind of problems and barriers does technical debt in software development create for improvement initiatives because the poor code base quality was often seen as the result of hack solutions and as the reason for not being able to improve several things. Related to this kind of research, it could be explored that what are the ways and benefits of paying back the technical debt in large-scale organizations involved in software-intensive product development.

Using the focus group method and the seven principles of lean software development as a method for driving the lean transformation in companies deserves further study. How this kind of an approach could be used in conjunction with transformation road mapping...
could be one focus area for research. Also, how to facilitate this kind of an approach in large-scale to provide meaningful results to be used to drive change could be interesting to study further.

As listed above, it is obvious that this research has provided good insight into the various possibilities for future research to take a look at. Also the possibility of conducting research on lean software development as a case study in companies with similar characteristics would be beneficial because it would enable analytical generalization. This means that the results can be extended to cases with similar characteristics.
References


## Moderator's Guide Group 1

**Date:** 10.06.2015  
**Moderator:** Riku Suomela  
**Total (mins):** 180  
**Focus Group session number:** 1  
**Total (hours):** 3

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Aids</th>
<th>Duration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td>Objectives of the study and the focus group; Practicalities: confidentiality, session rules; Emphasize that the participants opinions should represent the real situation;</td>
<td>Forms</td>
<td>5</td>
<td><strong>12:30</strong></td>
</tr>
<tr>
<td><strong>Current way of creating software</strong></td>
<td>Please write down what kind of challenges and problems there exist in the current way of creating software in your team / organization (2 to 3 items) Participants write;</td>
<td>Numerated Post-It's and pens</td>
<td>5</td>
<td><strong>12:35</strong></td>
</tr>
<tr>
<td><strong>Current way of creating software</strong></td>
<td>Participants explain;</td>
<td></td>
<td>20</td>
<td><strong>12:40</strong></td>
</tr>
<tr>
<td><strong>Practitioners’ understanding on lean (RQ1)</strong></td>
<td>How do you understand Lean in SW development?; Do you think that your current way of working has lean characteristics?; Participants write;</td>
<td>Numerated Post-It's and pens</td>
<td>5</td>
<td><strong>13:00</strong></td>
</tr>
<tr>
<td><strong>Practitioners’ understanding on lean (RQ1)</strong></td>
<td>Participants explain &amp; discuss</td>
<td></td>
<td>15</td>
<td><strong>13:05</strong></td>
</tr>
<tr>
<td><strong>Break</strong></td>
<td>Coffee time</td>
<td></td>
<td>15</td>
<td><strong>13:20</strong></td>
</tr>
<tr>
<td><strong>Lean SW Development principles (RQ2 &amp; RQ3)</strong></td>
<td>In this session four of the seven principles by Poppendieck &amp; Poppendieck will be explained and discussed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Create knowledge</strong></td>
<td>Explain the principle; o Explore and evaluate different solution alternatives before committing to one o Start with a minimum set of requirements and gather feedback of the implementation continuously o Kaizen: Continuous Improvement o Root Cause Analysis Opportunities and barriers?</td>
<td>Slideshow</td>
<td>5</td>
<td><strong>13:35</strong></td>
</tr>
<tr>
<td><strong>Create knowledge</strong></td>
<td>Participants write</td>
<td>Numerated Post-It's and pens</td>
<td>5</td>
<td><strong>13:40</strong></td>
</tr>
<tr>
<td><strong>Create knowledge</strong></td>
<td>Participants explain &amp; discuss</td>
<td></td>
<td>15</td>
<td><strong>13:45</strong></td>
</tr>
</tbody>
</table>
| Deliver fast | Explain the principle;  
|             | o Delivering working software as fast as possible (Shorter cycle times, small batches, limit work to capacity)  
|             | o Limit Work in Progress (WIP)  
|             | **Opportunities and barriers?** | Slideshow | 5 | 14:00 |
| Deliver fast | Participants write | Numerated Post-It's and pens | 5 | 14:05 |
| Deliver fast | Participants explain & discuss | 15 | 14:10 |
| Respect people | Explain the principle; Moving decision-making to the lowest possible level; empower people; encourage teamwork **Opportunities and barriers?** | Slideshow | 5 | 14:25 |
| Respect people | Participants write | Numerated Post-It's and pens | 5 | 14:30 |
| Respect people | Participants explain & discuss | 15 | 14:35 |
| Defer commitment | Explain the principle;  
|                  | o Scheduling irreversible decisions in the last responsible moment  
|                  | o Maintain your options when developing code  
|                  | o Remove dependencies in the architecture  
|                  | **Opportunities and barriers?** | Slideshow | 5 | 14:50 |
| Defer commitment | Participants write | Numerated Post-It's and pens | 5 | 14:55 |
| Defer commitment | Participants explain & discuss | 15 | 15:00 |
| Debrief | Sum up main discussion points for validity | Present notes | 10 | 15:15 |
| Wrap-up the session | Thank the participants and close the session; | | 5 | 15:25 |
| **End** | | | | 15:30 |
# Appendix B: Moderator’s Guide Group 2

<table>
<thead>
<tr>
<th>Moderator’s Guide Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date:</strong> 15.06.2015</td>
</tr>
<tr>
<td><strong>Moderator:</strong> Riku Suomela</td>
</tr>
<tr>
<td><strong>Total (mins):</strong> 175</td>
</tr>
<tr>
<td><strong>Focus Group session number:</strong> 2</td>
</tr>
<tr>
<td><strong>Total (hours):</strong> 2.916666667</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Aids</th>
<th>Duration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td>Objectives of the study and the focus group; Practicalities: confidentiality, session rules; Emphasize that the participants opinions should represent the real situation;</td>
<td>Forms</td>
<td>5</td>
<td><strong>12:30</strong></td>
</tr>
<tr>
<td><strong>Practitioners’ understanding on lean (RQ1)</strong></td>
<td>How do you understand Lean in SW development?; Do you think that your current way of working has lean characteristics?; Participants write;</td>
<td>Numerated Post-It’s and pens</td>
<td>5</td>
<td><strong>12:35</strong></td>
</tr>
<tr>
<td><strong>Practitioners’ understanding on lean (RQ1)</strong></td>
<td>Participants explain &amp; discuss</td>
<td></td>
<td>15</td>
<td><strong>12:40</strong></td>
</tr>
<tr>
<td><strong>Current way of creating software</strong></td>
<td>Please write down two things that are good and two things that are not good in your current way of creating software; Participants write;</td>
<td>Numerated Post-It’s and pens</td>
<td>5</td>
<td><strong>12:55</strong></td>
</tr>
<tr>
<td><strong>Current way of creating software</strong></td>
<td>Participants explain;</td>
<td></td>
<td>30</td>
<td><strong>13:00</strong></td>
</tr>
<tr>
<td><strong>Break</strong></td>
<td>Coffee time</td>
<td></td>
<td>15</td>
<td><strong>13:30</strong></td>
</tr>
</tbody>
</table>

**Lean SW Development principles (RQ2 & RQ3)**

In this session the participants will be given freedom of choice on prioritizing discussion on different principles. This decision to give the freedom of choice is based on the experience from the previous focus group session. Use slideshow for explaining each principle before discussion.

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Principle 1</strong></td>
<td><em>Create Knowledge</em>: Explain and Discuss <em>Opportunities and Barriers</em></td>
<td><strong>13:45</strong></td>
</tr>
<tr>
<td><strong>Principle 2</strong></td>
<td><em>Deliver Fast</em>: Explain and Discuss <em>Opportunities and Barriers</em></td>
<td><strong>14:00</strong></td>
</tr>
<tr>
<td><strong>Principle 3</strong></td>
<td><em>Respect People</em>: Explain and Discuss <em>Opportunities and Barriers</em></td>
<td><strong>14:15</strong></td>
</tr>
<tr>
<td>Principle 4</td>
<td>Defer Commitment: Explain and Discuss <strong>Opportunities and Barriers</strong></td>
<td>15</td>
</tr>
<tr>
<td>Principle 5</td>
<td>Eliminate Waste: Explain and Discuss <strong>Opportunities and Barriers</strong></td>
<td>15</td>
</tr>
<tr>
<td>Principle 6</td>
<td>Build Quality In: Explain and Discuss <strong>Opportunities and Barriers</strong></td>
<td>15</td>
</tr>
<tr>
<td>Principle 7</td>
<td>Optimize the Whole: Explain and Discuss <strong>Opportunities and Barriers</strong></td>
<td>If there's time left</td>
</tr>
<tr>
<td>Debrief</td>
<td>Sum up main discussion points for validity</td>
<td>Present notes</td>
</tr>
<tr>
<td>Wrap-up the session</td>
<td>Thank the participants and close the session;</td>
<td>5</td>
</tr>
<tr>
<td><strong>End</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>