Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Oskar Ehnström

Using lean to manage uncertainty in software development projects

A consulting perspective

Master's Thesis
Espoo, December 1, 2015

DRAFT! — October 13, 2015 — DRAFT!

Supervisor: Professor Marjo Kauppinen, Aalto University
Advisor: Suvi Uski (?)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Oskar Ehnström

Title:
Using lean to manage uncertainty in software development projects A consulting perspective

Date: December 1, 2015 **Pages:** vii + 23

Major: Software Engineering and Business **Code:** T-76

Supervisor: Professor Marjo Kauppinen

Advisor: Suvi Uski (?)

The current literature on lean software development is derived from the literature on lean manufacturing. These principles can be applied as various practices in software projects depending on the context. The principles hold well for the general domain of software development.

Lean enables dealing with uncertainty, which has been found to be crucial when dealing with changing requirements. Software development often deals with uncertain requirements as well as new and unfamiliar technology. Whether lean could be used to deal with uncertainty in both requirements and technology has not been studied.

This thesis studies whether lean principles work as such for projects where the technology domain is unfamiliar to the project team. The study finds that the principles hold for projects facing uncertainty in both requirements and technology.

The study uses one software project as a case study. As such, general conclusions about the subject can no be drawn. However, the results suggest that lean could be a useful tool in handling uncertainty caused by unfamiliar technology in software projects.

Keywords: lean, lean software, lean software project, service creation, agile, LSC

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Oskar Ehnström		
Työn nimi:			
Päiväys:	1. joulukuuta 2015	**Sivumäärä:**	vii + 23
Pääaine:	Ohjelmistotuotanto ja liiketoiminta	**Koodi:**	T-76
Valvoja:	Professori Marjo Kauppinen		
Ohjaaja:	Suvi Uski (?)		
Asiasanat:			
Kieli:	Englanti		

Aalto-universitetet
Högskolan för teknikvetenskaper

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV
DIPLOMARBETET

Utfört av:	Oskar Ehnström		
Arbetets namn:			
Datum:	Den 1 December 2015	**Sidantal:**	vii + 23
Huvudämne:	Programvaruproduktion och affärsverksamhet	**Kod:**	T-76
Övervakare:	Professor Marjo Kauppinen		
Handledare:	Suvi Uski (?)		
Nyckelord:			
Språk:	Engelska		

Acknowledgements

TODO: Thank people here

Espoo, December 1, 2015

Oskar Ehnström

Contents

Chapter 1

Introduction

- Lean dictionary definition
- Where does lean come from?
- Why was lean manufacturing needed?
- Why was it successful?
- Specific reasons = principles
- How can these be used to battle uncertainty?
- Lean Startup
- Unknown environment
- Does this translate to unknown technology
- If it does, it could be used for competitive advantage and better predicitons

1.1 Scope

The scope of the literature review will be existing literature on lean software projects, comparing these to find similarities if there are any.

Scope of the empirical study will be one lean software development project. The project will be studied from the point of view of developers, customers and end-users. The thesis is limited to one case study.

1.2 Structure of the Thesis

This section presents the structure of the thesis.

!FIXME **Write these a bit smoother once the structure is done** FIXME!

Chapter 2 presents the origin of the lean philosophy. It goes through the development of lean principles from manufacturing to software development.

Chapter 3 covers the existing literature on lean software development. In this chapter experiences of previous lean software projects are analyzed and compared. This is done in order to find some common trends or best practices to use in projects. These common trends and best practices are compared in order to later compare them with the findings of the empirical study, which are presented in chapter 6.

Chapter 4 goes through the methods used in the empirical study. The participants and their roles are presented as well as the practical arrangements regarding interviews.

Chapter 5 presents the results of the empirical study. The data is analyzed and the results of that analysis are presented.

Chapter 6 discusses the results gathered from the empirical study and their implications. This is done by analyzing the relationship between the data gathered from the empirical study and the literature review presented int chapter 3.

Chapter 7 presents the conclusions of this thesis and suggestions for further research.

Chapter 2

Background

2.1 The origin of lean thinking

!Fixme **This section will be about the history of lean and how it came to be applied to software engineering** Fixme!

Even though the traditional wisdom is that the Japanese car manufacturers had a significant advantage over western competitors due to the lean methodologies they implemented there is some controversy over whether this was in fact the case. Dybá & Sharp argue that by examining the facts and taking automation into account the Japanese did not have a superior organizational advantage. [1]

2.2 Lean versus Agile

!Fixme **This section will discuss the differences between lean and agile** Fixme!

!Fixme **Lean should be thought of a set of principles rather than practices. This article has some excellent points and trends to talk about.[6]** Fixme!

Chapter 3

Literature review

This chapter presents the existing literature on lean software development. The methods used for gathering academic sources is presented, as well as a general overview on the subject. Finally existing literature on past lean software development projects are compared to the general principles of lean software development and to each other.

3.1 Literature overview

This section presents the methods used to gather material for the literature review.

Google Scholar was the primary source for material on lean and lean software development. Searching the numerous databases in Google Scholar with the keywords mentioned in figure 3.1. Once some suitable primary articles had been found these could be used to find related articles. By utilizing the related articles feature on Google Scholar more related articles could be found. The sources of known articles also led to other articles on the subject and especially to heavily cited articles that provide the foundation for many other articles. Searching for articles written by the authors of known articles also led to other articles that dealt with the subject. Conferences and journals that included some known articles also proved to include other similar articles and were a good source of material.

3.2 Lean software principles

One book in particular has been the foundation of many lean initiatives of various kinds. This is the work by Poppendieck & Poppendieck in “Lean

Search term
lean software development
lean software management
lean project management
digital service creation
lean

Figure 3.1: Keywords used to find sources

Software Development: An Agile Toolkit". In this book the authors motivate why lean works in software development and translate some of the common aspects of lean manufacturing into the language of software development. They also present seven principles of lean software development. These are adaptations and expansion of the principles developed for lean manufacturing.[7]

3.2.1 Eliminate waste

Eliminate waste is the fundamental principle of lean. Waste in this context is understood as anything that does not produce value to the customer. This may seem like a clear definition, but once one starts measuring what is actually producing value and what is not the results may be surprising. In manufacturing inventory is considered to be waste. Ideally products should move from one stage to the next immediately. In software engineering one equivalent of inventory is unfinished features.[7]

One of the main reasons why inventory is considered waste is that inventory may contain defects that have not yet been discovered. These defects become much more expensive to fix later in the process. Defects are discovered much faster in the next steps of the process when batch sizes are small and inventory is minimal. In software engineering inventory often manifests as features waiting for testing or other forms of approval. Inventory in the form of features stuck at some crucial point in the system may also be blocking other, more important features.[7]

Learning to see waste can be challenging when unfamiliar with the concept. For this reason Poppendieck & Poppendieck have gathered together a list of the seven wastes of software development. This list is based on the seven wastes of manufacturing by Shigeo Shingo[7]!Fixme **Does this need its own reference?** Fixme!. This list is presented in table 3.1.

Eliminate waste is one principle that is present in all of the case studies presented in this paper [2][4][3][9]. This seems to indicate that this is indeed

The Seven Wastes of Software Development
Partially done work
Extra processes
Extra features
Task switching
Waiting
Motion
Defect

Table 3.1: The seven wastes of software development as defined by [7]

considered to be the most important principle of lean.

3.2.2 Amplify learning

Amplify learning is the second principle presented in the book, and focuses on quality and learning[7]. This principle was later redefined by Poppendieck as "learn constanstly"[5]. The principle of learning is crucial in lean, as the whole process is based on the aforementioned concept of removing waste. For this to be successful waste needs to be identified and this skill has to be learned. The book presents several tools to help facilitate learning.

Many of the tools presented are familiar from the world of agile software development. Feedback is a natural and effective part of learning. For this to be an effective tool in practice there needs to be a feedback cycle and effort to gather feedback. The book also mentions that software projects are often ill structured problems where a mentality of trying something first then fixing it works better than a predetermined plan.[7] This ties closely to the build-measure-learn cycle proposed by Ries [8]. Another way of looking at is is seeing the software context as a chaotic system where a clear solution is not known.!Fixme **source?** Fixme!

The authors also mention iterations, a concept central to agile development. Iterations enable learning as knowledge about the process and project is refined each iteration. [7]

A couple of tools that are probably not as familiar to those familiar with agile methodologies are synchronization and set-based development. Synchronization is closely related to continous integration and set-based development explores multiple options simultaneously before committing to one solution.[7]

3.2.3 Decide as late as possible

Deciding as late as possible is about keeping several option available until the last responsible moment to make a decision. The reason is the same for lean as it is in agile: there will be changes, so the right thing to do is prepare to be able to handle change.[7]

Lean principles recognize that it has hard to understand a problem perfectly the first time and plan accordingly. Keeping several options available while learning about the problem and trying different approaches increases the chance of having a working solution when a decision has to be made. [7]

The book also points out that deciding as late as possible is not about avoiding decisions until it is too late. A decision should be made at the last responsible moment.[7] Of course, it may not be easy to know exactly when that moment is, but it is still better than not deciding or locking oneself to one solution that turns out to be the wrong one.

This chapter of the book also mentions several practices that are generally considered good development practices. Separation of concerns, interfaces, and modules are mentioned as some examples of practices that enable deciding as late as possible.[7] All of the examples mentioned in the book are ones that could be considered universal good practices for software development. It follows, that lean does not conflict with agile, or indeed any other development methodology, but rather encourages good praxis.

Empowerment is also mentioned as a way of deciding as late as possible. In this context empowerment is about teaching the people working on a project to how to make decision. The idea is that the people working on the project are those who know the needs of the project and can make the best decisions about how to proceed. By allowing the team to make the decision it is possible to avoid overhead and get the best possible information available. There is no need for management to get involved, which frees up management resources for other tasks.[7]

3.2.4 Deliver as fast as possible

Delivering as fast as possible is a lean practice as well as an agile one. In lean the idea is that when you deliver fast you get feedback fast and can react to the feedback. To deliver fast Poppendieck & Poppendieck suggest a few tools to use. The first, and perhaps most fundamental, tool to deliver fast in lean is limit WIP. By settings hard boundaries on the amout of work in the system at any given point the system is encouraged to seek out bottlenecks and resolve them. Reducing WIP also leads to concrete timesavings in the form of reduced switching times and the drop in utilization that these cause.[7]

The pull system in such a way that work is not pushed to the next stage, but instead each stage pulls work from the previous stage. This system is used to make sure no part of the process becomes overworked. It also helps people feel empowered, as they know their tasks and can work independently by choosing their tasks from the list of work available from the previous stage.[7]

This chapter also presents queuing theory and how it fits with the idea of lean and delivering fast. The theory is concerned with cycle time, the time it takes from an entity to entering the process to the time it exits. This time needs to be short in order to be able to deliver fast, and for it to be short there can be little queues and variability in the system.[7]

The authors also point out that sometimes it is enough to calculate how much time a new tool will save compared to the cost of acquiring the tool. This is because delivering fast might have an impact on how valuable the delivered product is.[7] For example, being first to market might allow you to ask for a premium price that would make the tool wort the cost after all.

3.2.5 Empower the team

Empowering the team is about motivating the team to do great work. If the team feels they have a purpose and are able to work towards it they can be more productive compared to being told what to do. Access to the customer is mentioned as a positive factor for both motivating and empowering the team.[7] This concept of empowering the team is closely related to the concept of self organizing teams that agile promotes. The self organized team moves towards a common goal and organizes themselves in the best way to achieve that goal.

The lean approach promoted by Poppendieck & Poppendieck mentions the "master developer" as a tool to be used to empower the team. This is the concept of a experienced developer who keeps the overall picture in mind.[7] This approach stems from the concept of the master engineer at Toytota[7], but is not commonly used as a concept in agile. It may be that the concept of self organizing teams and scrum masters have eclipsed the master developer.

3.2.6 Build integrity in

The concept of integrity is in this context, if not completely then approximately, the same as quality. Software quality seems like the preferred phrasing these days and if integrity is considered the same as quality a lot of similarities with the lean and agile approaches emerge.

The authors mention model driven design as a tool to achieve integrity. They describe it as a method where a model is construction on which all development is based. This model is understandable by both the customer and the development team.[7] This concept is very close to the agile approach of acceptance testing. Where the customer is asked to test the product in order to validate that a feature was implemented as intended.

Another tactic to achieve integrity presented by Poppendieck & Poppendieck is to use proven components.[7] One possible way to achieve this in daily development would be to use open source components that have been used elsewhere and are battle tested, so to speak.

Refactoring and testing are also mentioned as tools to build in quality.[7] Both are widely accepted good practices and should be a rule rather than an exception in any software project.

The main takeaway from the lean concept of building integrity in, as presented by Poppendieck & Poppendieck, is that they tools they present and advocate for are not that different from the tools used today in software development to achieve good quality. Lean software development advocates for sound engineering principles.

3.2.7 See the whole

See the whole, again, emphasizes the lean concept of flow and optimizing the whole process rather than individual steps. In fact, local optimizations are mentioned as one specific problem that may seem like added value, but are in fact adding waste to the overall process. One solution to this is to use aggregate data. By using data that is not tied to a specific measurable entity, the focus stays on the overall picture.[7]

The authors also point out that it is often hard to see the real cause of a problem. To overcome this they suggest the method of the five whys. Ask “Why?” five times, and you are more likely to find the root cause of a problem and not only the symptoms.[7] This method is heavily promoted at Futurice, and has led to successfully solving customers’ root problems instead of addressing symptoms.

The whole may also be extended outside ones own organization. Including vendors and suppliers in the overall pictures results in possibilities to extend the lean methods further than ones own organization and claim the benefits. This may of course be difficult, and the book goes through several forms of contracts with their pros and cons. The foundational idea in this relationship is trust. When suppliers, vendors and customers can trust each other waste is minimized and value is maximized.[7]

3.3 Past case studies

This section presents the existing literature on lean software projects.

3.3.1 Timberline Inc.

The case study of Timberline Inc. appears to be one of the first case studies done on lean software development methods. In this study the authors study how a traditional software development company used lean principles in order to streamline work and cut waste. The authors were also interested in how companies involved in both manufacturing and software could match the already implemented lean processes of manufacturing to software development. At this time software development was not generally thought of as lean or iterative, but it was clear that software was becoming a larger part of manufactured products.[3]

The focus of this lean transformation was the concept of 'takt', the tempo of development. The development team was forced to switch between tasks as too much work was being pushed into the system, causing the amount of work in progress to increase. Incomplete code as well as time spend on switching tasks are sources of waste in lean software development. This switching back and forth, also called thrashing, was mitigated by splitting projects into smaller pieces called 'kits'. Smaller units of work and a WIP limit effectively causes the system to flow more smoothly.[3] This concept of smaller batch sizes is essential for any iterative process as well as the fast feedback cycle used in lean.

Timberline also used the concept of a daily stand-up meeting familiar from agile. They also moved team members around to form co-located teams. This was mentioned to cause some dissatisfaction in the well established organization, but the challenges were overcome.[3] This concept of co-located teams is a core concept of agile for the same reasons it is important in lean software development. Having a short as possible feedback cycle is crucial to catch defects as soon as possible, before they become costly.

The existing processes in the organization caused problems for the movement of people to those areas of the production flow that needed more resources. The reason for this was simple legacy and history. Once these outdated processes were identified they could be improved easily.[3] These habits and old processes may often persist only because there is no radical change in the environment. Adopting lean processes with an open mind will, at the very least, allow the organization to take a critical look at the existing

processes.

The study also points out the importance of creating a learning organization by gathering information and using it to improve the process. The study does not go into detail as to how this was accomplished. The test used for how well information was displayed was that a person unfamiliar with the work could walk into the project area and see the current state.[3]

Self-managing teams were also a result of gathering data and using it for decision making. The positive results mentioned in the study were reduced management costs[3] but another aspect of a self-managing team is likely to be empowerment. The motivation of the team may very well rise considerably when they feel empowered to make their own decisions. Empowerment is also one of the seven principles persented by Poppendieck & Poppendieck [7].

The clear conclusion of the study was that lean techniques do transfer into software development. Processes were clearly improved. Extreme examples of improved processes included one in which only 1.4% of steps added value before the move to lean. After implementing lean the time to fix defects during the development cycle dropped 65%-80%, among other improvements. Customer satisfaction was also observed to be "overwhelmingly positive" and was speculated to be because of iterative development.[3]

This study, like others, points out that the move to lean has to be a company wide effort.[3] This is tightly related to the concept of optimizing the whole. If the whole organization is not on board with the lean transformation, then the value added by it can only be limited to part of the organization at best.

3.3.2 BBC Worldwide

This study followed the introduction of lean practices in a nine person team working for the BBC Worldwide organization. The team worked on various tasks for customers inside the BBC and included management, development and testing roles.[4]

The study focused a fair bit on lead time. That is, the time from a customer request to the time the requested entity is delivered. Variance in lead time was something that was to be reduced, as low variance makes for a more stable and predictable process. By reducing variance and decreasing the size of units of work, the team was able to reduce cycle times and achieve a greater number of small deliverables.[4] This type of increase in number and decrease in size can enable faster iterations and allows a team to build the right thing more quickly.

Building in short iterations also reduces risk, as there is less unfinished code waiting. This reduces both the risk of building the wrong thing as well

as the risk of bugs in the code. The study noted a practical example of this by stating that customers were able to evaluate a "...tangible product rather than just progress reports". The data also indicated that there were fewer bugs in the code and these bugs were fixed faster.[4]

A familiar practice from agile, the daily stand-up, was used by the team. However, the study points out that the stand-up focuses on the tasks and data and not as much on the people, according to the study, is the custom in agile.[4] This could be a misunderstanding of the ideas of agile by the authors of the study. Both the agile and lean ways of conducting stand-ups focus on knowledge sharing in order to facilitate a smoother development process. Badly implemented ways of doing this may of course affect either implementation and begin to focus more on assigning blame and measuring people, which is usually seen as unhealthy. The study also pointed out that the stand-ups were not meant for team members to give reports of their work. The most efficient way to conduct the stand-ups was for each team-member to highlight problems they had encountered so these could be addressed and the flow would not be interrupted.[4] This is also the idea behind the agile stand-ups, but it can be misunderstood.

Performing these daily stand-ups in front of the Kanban boards used by the team served a special purpose. By analyzing the placements of the tasks on the board it was easy to visually see bottlenecks which interrupted the flow of development. Eliminating these bottlenecks would then be one way for blocked team members to do meaningful work while not breaking their WIP limit, as specified by the lean principles.[4]

The type of work the team performed did not change, but the way they handled the requested pieces did. The team focused on splitting requests into small pieces with, what they called, Minimal Marketable Features.[4] This idea is closely related to the Minimum Viable Product idea presented by Eric Ries. [8] Both are in essence the same approach of delivering maximal value to the customer with minimum effort in order to then learn and adapt.

The study also pointed out that lean practices are not an alternative or replacement of professional software engineering practices.[4] This supports the idea that lean is a set of principles, rather than practices. Challenges related to the new, lean, process was, according to the study, often due to constraints with the existing organization and their ways of working compared to the team using lean. The study listen this as a reason why lean is more about a cultural shift than it is about implementing certain practices. [4]

3.3.3 Electrobit

Electrobit is a Finnish provider of wireless embedded systems. The case study was conducted in 2010 and the organization had used agile practices since 2007. The case study followed how some key performance indicators (KPI) changed as the organization started using lean practices. [9]

The study was particularly focused on how lean and agile practices can be combined in software development. The authors focused on elements that characterize the combination of lean and agile. They were also interested in what challenges the combination presents, as well as which elements of the combination were easy to implement.[9] As agile has only appeared to increase in popularity in the software development business this study is a good reference on how lean and agile can be combined in order to produce software. The hardware related business also presents some unique challenges compared to purely software based models of lean and agile.

The study found that the move to lean had indeed influenced practices of the company. Discussions with employees revealed that the lean principles had expanded concepts like reducing waste to be considered throughout the organization. In comparison, previous practices had left these responsibilities largely on the product owner alone. However, the change from agile to lean was described as "an incremental improvement in which Agile is not abandoned when Lean is adopted."[9]. This is quite natural, as agile practices focus more on the software development work whereas lean takes a more holistic approach to the whole value chain.

When the subject was speed and flexibility the discussions with the subjects of the study focused on the importance of short lead-times and the ability to cope with change.[9] These are important questions business-wise, as maneuvering fast and responding to change can be the deciding factor in a fast paced environment like software development. Responding to change is one of the things both agile and lean aim to enable in order to cope with chaotic systems.

Eliminating waste was seen as an aspect specifically related to lean principles that had not been incorporated in the earlier, agile, practices. Tightly related to that, seeing the whole also brought a more holistic approach to the company compared to the agile mindset before the change. When discussing specific practices, minimizing work-in-progress (WIP) was found to be easy to motivate and understand as a way to reduce waste.[9] This becomes clear when one thinks of unfinished code as being the software equivalent of inventory with possible bugs and unnecessary or unwanted features.

Electrobit's ways of working also enabled them to have short feedback cycles. This is a critical component for the ability to adapt and respond

to change quickly. They also handled uncertainty by continuous learning. Estimations were small but accurate, which is the typical way for agile estimations to work. Another aspect that related to both agile and lean was that participants in the study pointed out that delaying decision making should not affect the release of the software.[9] This borrows from agile as it does not allow the schedule to be delayed, but recognizes that lean principles call for delaying a decision until it has to be made in order to keep all options open.

Lean principles emphasize "perfection" or "learn constantly", at Electrobit they found that Kanban provided an added value as a process because of its ability to visualize queues and enable finding the root cause of problems [9]. Agile also focuses on continuous improvement, so the change towards lean was likely a natural evolution towards organizational learning from the more team focused approach of agile.

Learning was enabled by organizational transparency. This was the most stressed element of the discussions with the participants of the study. Transparency enabled knowledge sharing and enabled visibility on all organizational layers and in all directions. [9]

Finally, participants chose to mention the people factor of software development. This is very much related to agile, but has its place in lean principles as well in the form of "engage everyone" as presented by [7].

The study presents some challenges that Electrobit encountered regarding lean principles. Flexibility of the whole value stream was a challenge. Teams also perceived that they were unable remove waste even though they had identified it due to complex project set-ups. Long feedback loops were still an issue due to challenges in involving management and third parties into the development process. [9] These challenges are familiar from traditional lean manufacturing and agile software development, which could mean that lean software development was not a silver bullet in this case, even if it did improve the overall situation.

!Fixme **TODO: compare more thoroughly with other cases and poppendieck** Fixme!

3.3.4 Two Case Studies

In "Lean Software Development: Two Case Studies"[2] author Peter Middleton sets out to study whether lean principles can be applied to software development. This study appears to be one of the first studies that tackles this question and predates the work of Poppendieck & Poppendieck published in 2003.

The study presents the foundation of lean principles and how they might

be applied to software development. The most important result, however, is the experiment conducted by Middleton to study the effects of applying lean methods on a traditional software process. In this experiment, two small teams in a large organization were selected to try lean practices in their daily work.[2]

The traditional process was first streamlined somewhat to enable the introduction of lean principles. Once the new process was stable, lean principles were introduced. The implementation focused most on aiming to reduce waste by stopping the process once a defect was found. This initially slowed down both teams, as team members were not allowed to work on other tasks while the issue was resolved. This was done to limit their WIP. Once the teams learned to see defects and address them more quickly overall progress improved compared to the traditional process.[2]

Although the study is very limited, focusing on two small teams for a short period of time, it is able to highlight some of the most prominent organizational challenges. In this particular organization the hierarchical structure of the teams introduced friction in reporting issues, which is an essential part of lean. The hierarchy also fostered a culture where people needed to take jobs they had little aptitude for in order to advance in their careers. One manager also felt that they were securing their job by not sharing information. This could have been a result of the organizations less than ideal policies on learning, which were mentioned as a challenge for the application of lean. Finally, some aspects of the lean process were hindered by third parties inside the organization who were unable to deliver the required quality.[2]

An overall conclusion of the problems uncovered in the study was that problems were often a result of organizational challenges, and the issues with quality were, in fact, the symptom of deeper problems. Finally, the study also concluded that "no inherent reason has been found to suggest that lean techniques cannot be used in software process."[2]. This might have influenced others to try to replicate these results in larger studies.

!Fixme **TODO: compare more thoroughly with other cases and poppendieck** Fixme!

3.3.5 Others

!Fixme **Mention other studies?** Fixme!

	Timberline Inc. (3.3.1)	BBC Worldwide (3.3.2)	Electrobit (3.3.3)	Two Case Studies (3.3.4)
Eliminate waste	Strong	None	Strong	Strong
Flow	Strong	Strong	Strong	None
Learning	Strong	Strong	Strong	Strong
Empowerment	Strong	None	Strong	Strong
Pull	None	Strong	Strong	None
Deliver fast	Some	Strong	Strong	None
Quality	Some	Some	Some	Strong

Table 3.2: Focus areas of the case studies

3.4 Comparison

One thing is common amongst all the case studies: They are done at companies producing their own products using a known technology stack. A software consultancy company could face a host of unknowns regarding the business context of their customers as well as the technology that is used. However, a lean and agile partner could be exactly what a traditional behemoth needs to stay competitive in a fast paced business.

3.5 Research problem and question

This chapter will end with the research problem and questions.

The research problem is defined as follows:

What are the commonly accepted and used lean software development practices and how do they change when working with new and unfamiliar technology?

To investigate this problem three research questions have been set up in table 3.3.

Question	Literature review	Empirical study
What are the currently available best practices for lean software projects?	x	
Which of these best practices need to be adapted when working with new technology?		x
How do these best practices need to adapted?		x
Which best practices remain valid?		x

Table 3.3: Research questions and their respective sections

Chapter 4

Methods

In this section I will describe how I conducted the empirical study.

- Interviews
  - 2-3 project members
  - 2-3 customers
  - 2-3 end-users
  - 1-1.5 hours each (5-10min for end-users)
- Semi-structured interviews

Chapter 5

Results

In this section I will present the results of the empirical study. This section contains the raw data of the conducted interviews. This section does not analyze or compare the results with the literature.

Chapter 6

Discussion

Here I will discuss how the findings from my empirical work relate to the literary review. What are the similarities and differences when comparing what the literature says and what the interviews showed.

6.1 Lean Service Creation

!Fixme **Lean Service Creation is about build measure learn and create something new. Tie this into the problem and focus on the new part. Using new devices maybe?** Fixme!

Lean Service Creation is based on the ideas of "The Lean Startup" as described by Eric Reis in his 2011 book.[8]

!Fixme **How do I get a source for this? Interview?** Fixme!

Simple rules to guide decisionmaking: 3x2 at Futurice.

Chapter 7

Conclusions

Here I mention the most important findings of the discussion section and the literary review section.

I also point out how this research can be used in the future and what it's limitations are. (e.g. only one case study)

2 pages

Bibliography

[1] Dybà, T., and Sharp, H. What's the evidence for lean? *IEEE Software 29*, 5 (2012), 19–21.

[2] Middleton, P. Lean software development: Two case studies. *Software Quality Journal 9*, 4 (2001), 241–252.

[3] Middleton, P., Flaxel, A., and Cookson, A. Lean software management case study: Timberline inc. *Extreme Programming and Agile Processes in Software Engineering* (2005), 1–9.

[4] Middleton, P., and Joyce, D. Lean software management: Bbc worldwide case study. *IEEE Transactions on Engineering Management 59*, 1 (2012), 20–32.

[5] Poppendieck, M. Lean software development principles. *October* (2010), 1–2.

[6] Poppendieck, M., and Cusumano, M. a. Lean software development: A tutorial. *IEEE Software 29*, 5 (2012), 26–32.

[7] Poppendieck, M., and Poppendieck, T. *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

[8] Ries, E. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Random House LLC, 2011.

[9] Rodriguez, P., Partanen, J., Kuvaja, P., and Oivo, M. Combining lean thinking and agile methods for software development: A case study of a finnish provider of wireless embedded systems detailed. *2014 47th Hawaii International Conference on System Sciences* (2014), 4770–4779.

Appendix A

Interview questions

Here goes the questions from the interviews.