Preventing challenges in Lean Startup methodology

- from a software engineering perspective

By

Nicklas Warberg
Nicolai Nørbjerg Thorup

A master thesis written for:
MSc in Business Administration and Information Systems (E-Business)
at
Copenhagen Business School, Denmark

CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

2015

Supervisor: Stefan Henningsson

Abstract

The field of entrepreneurship is recognized as being a key factor in economic development, especially in the IT sector. The number of new IT companies has increased dramatically in the last decade with many success stories like Facebook, Skype and Google. However, these success stories does not reflect the general situation in the IT startup market. Recent research shows that the success rate is alarmingly low with the majority of venture-backed startups does not return inventors' capital. This extensive problem is what the Lean Startup Methodology (LSM) tries to solve by introducing a set of new techniques and principles that will supposedly provide a completive edge in the startup market. LSM has received a lot of attention in recent years with its new and controversial approach. Even though LSM has been adapted by many companies it has received criticism for being too detached from modern software development practices. The purpose of this thesis is to identify specific technical challenges and create preventative guidelines that will create a better foundation for new companies using LSM.

A set of technical challenges associated with LSM were identified. The challenges focuses on three specific areas. First, it was discovered that the LSM process devalues proper architecture in the software. Second, LSM creates unnecessary waste in the software and third, the LSM process could hinder innovate solutions in the software.
Three corresponding guidelines were created in order to prevent these challenges. The guidelines were evaluated in a real life startup company. The first guideline stated that you should make important decisions as late as possible. It was discovered that a number of unnecessary features were never implemented as a direct result of this guideline. The second guideline was a new type of software architecture specifically designed for LSM. It was discovered that the software became more compatible of handling changing requirement due to this guideline. Finally the third guideline stated that you should focus on innovation as a development activity. It was discovered that numerous ideas for possible solutions were created because of this guideline.

Table of Contents

List of figures

1) Introduction

This section introduces the research problem along with the motivation and research question for this master thesis.

1.1 Background

Entrepreneurship is one of the main drivers for economic growth around the world. New innovative ideas create competition in the marketplace thus speeding up structural economic changes (EC, 2013). Especially entrepreneurship in the IT sector has seen rapid growth within the last decade. When taking a look at today's most valuable global brands according to the Millward Brown's index, the top five consists of four technology firms; Google, Apple, IBM and Microsoft. In general, technology companies represents nearly a third of the total brand value - an increase of 16% since 2006. Companies like Skype (founded 2003), Facebook (founded 2004) and Dropbox (founded 2007) are all success stories giving examples at how fast IT-companies can potentially grow (Brown, 2014).

The evidence clearly shows that even small companies have the potential to create a billion dollar enterprise with relatively limited resources - however, these success stories are far from the norm. Every day new entrepreneurs are starting their own businesses. Most of them have (in their mind) an amazing idea for a product or service but lack the necessary tools for building an enduring business. Newer research done by Shikhar Ghosh, a senior lecturer at Harvard Business School, shows that 75% of all venture-backed startups in the U.S. do not return investors' capital (WSJ, 2012). In fact, 25% of new startups fail completely after only one year (Shane, 2010).

Why is it that so many entrepreneurs fail? Researchers have over the years tried to analyze the different factors, conditions and characteristics in successful startups in order to discover what it takes to create a successful business. Despite the fact that these characteristics have been analyzed and recommendations have been presented, real conventions for usage are not clear (Mcclelland, 1987; Hornaday 1971).

Recently, a new and controversial approach has emerged on how new startups should be managed and has already received a considerable amount of attention from both practitioners and educators in the field (Nobel, 2011). This new breed of literature is known as the Lean Startup Methodology (LSM) with Steve Blank as the pioneer of the term (Blank, 2006). LSM has a strong focus on actual execution and active suggestions for startups. The term was promoted by the Silicon Valley business visionary Erik Ries with his book "*The Lean Startup*" (Ries, 2011).

LSM revolutionizes the startup process by disregarding many of the traditional elements usually seen when trying to build a startup. When already established businesses launches a new product, it typically starts with a comprehensive business plan including a 5-year forecast. Therefore, it would be tempting to apply these principles to a startup business as well. However, Eric Ries believes that this will simply not work because new businesses work with extreme uncertainty. You do not know exactly who your customer base is and what your product should be. This uncertainty means that it becomes impossible to predict the future and therefore old management methods are not up to the task. Instead, LSM focuses on experimenting rather than detailed planning, customer feedback rather than intuition and iterative implementation over a full feature launch up front. (Blank, 2006; Ries, 2011)

The thought behind LSM is to incorporate the customers in all phases of the development to get valuable feedback in time for radical changes to be applied. At the early stage, you need to test if your idea is viable in the current marketplace. By including customers early on, you can validate whether or not your idea is sustainable or if it needs to be revised, this is what Eric Ries describes as validated learning (Blank 2006; Ries 2011).
In general, LSM focuses on constant learning by trial and error in order to align the specific product with to the actual market need thus creating the best chances of survival and sustainability.

Even though LSM sounds very promising for startups, a number of critiques has arisen over the years since it was introduced. Especially the lack of synergy with software development has received a considerable amount of attention due to the enormous development in the digital sector. With these problems, IT-startups are not able to utilize the full potential of LSM which can ultimately inhibit the chances for success.

1.2 Motivation

The Lean Startup has become one of the biggest buzzwords in recent years, not only for entrepreneurs but business people in general. The book "*The Lean Startup*" by Eric Ries is currently the number one best seller on Amazon in the category "*New Business Enterprises*" and has received critical acclaim (Amazon, 2015). As mentioned, LSM utilizes iterative product development with a focus on experimental learning. New ventures should quickly create a "*minimum viable product*" (MVP), which is a version of the final product who has only the minimum set of features in order to get feedback from the customers. Then, using that empirical data, decide to either persevere with the product or revise the hypothesis, which is called pivoting. A pivot is defined as a *"structured course correction designed to test a new fundamental hypothesis about the product, strategy, and engine of growth" (Ries, 2011, p.169).* When you pivot, you are essentially changing the direction you want to go while still maintaining the same foundation. An example of a pivot could be YouTube. YouTube started in 2005 as an online video dating site called "*Tune in Hook Up*". When the site failed to gain traction, they decided to go in another direction and focused on simply sharing videos online, which quickly became the world's biggest video sharing site. (The Guardian, 2009).

This iterative process of constantly revising your product makes sense seen from a business perspective when facing such extreme uncertainty. However when dealing with IT startups, it is important not to neglect one of the most essential resources - the software developers. The case of YouTube illuminates how you can completely change direction in how you approach the market but still keep your core foundation both from a business and technical point of view. Even though this change was quite considerable, smaller pivot might seem trivial to business people but can still affect the software dramatically. A small change in business strategy can easily result in a huge change in the code, which could result in a bunch of code being scrapped and an increase in development time (Bass et al, 2003). As Eric Ries describes on his blog he faced this exact problem when creating his own startup:

"*It was incredibly hard for me to throw out working code. [...] I was stuck between a rock and a hard place. Leading up to a pivot, each cycle, despite our best efforts, the metrics weren't good enough.*" (Ries, 2009)

The quote originates from the article *"Pivot, don't jump to a new vision"* by Eric Ries where he describes his own experiences as a software developer working in an IT startup. After a usability test, the team discovers that their product was not aligned with the market need and therefore they needed to pivot. However, in their case it was necessary to throw away working code even though the core functionality (an instant message client) remained the same throughout the whole process.

Whether it was due to lack of architecture or some other factors he does not mention but recently the LSM has received criticism for devaluating architecture in the software. Since you do not have time to develop the whole product but just the MVP you will not make time to invest in software architecture according to Michael Sharkey, the CEO of Bislr. A lack of well defined architecture in software can easily make small changes much more time consuming and completely hinder the scaling possibility (Venturebeat, 2013).

An example of a startup company that used architecture as a competitive advantages could be the case of Evernote. Many companies once competed for the same market as Evernote. A company called Catch even came complete with a colorful user interface. However, Evernote quickly became the dominant player on the market by taking advantage of their architecture and creating a platform that allowed independent developers to build on top of their existing product. Instead of focusing on enabling people to capture, store, and retrieve their Internet memories themselves, they realized the crowdsourcing potential, which was possible since Evernotes had such a strong focus on architecture early on (Venturebeat, 2013).

However, lack of architecture is not the only technical challenge startups have faced when using the LSM. Recently, we have seen research indicating that the technical complexity is not appropriately considered within the LSM. These software related problems can have very negative consequence for the company and potentially hinder the success of the startup because of increased development time or even inferior products. Therefore, it is very important to consider how these challenges can effectively be prevented in order create a better foundation for software startup and ultimately increase the success rate in these kinds of companies (Bosch et al., 2013; Venturebeat, 2013).

As business people with an educational background in software development our goal is to illuminate where in the LSM the technical challenges are overlooked and how they can be prevented in future IT startups. By creating synergy between the development- and business process it can help new entrepreneurs adapt their product or service to what the target audience actually wants.

1.3 Research question

We want to create a better foundation for success in IT startups by aligning the software development approach with the LSM. If the technical aspect has already been considered when starting a new business it is possible to can avoid some of the technical challenges one often faces in such an environment.

Our goal is to illuminate elements of LSM that developers have to be specifically aware of and provide technical guidelines in order to prevent challenges within LSM.

We will achieve this by first analyzing how the technical complexity is considered in the LSM according to different literature in this area. More specific, we will identify a set of challenges arising in the software development process when LSM is applied.
Based on the challenges identified by the literature we will provide concrete recommendations on how to prevent some of the larger threats software developers face when using LSM.

This leads us to the research question for this thesis.

Research question:

"How can you create preventative guidelines for the technical challenges one faces in the Lean Startup Methodology?"

1.4 The case company

This master thesis is written in collaboration with Flopfile. Flopfile is a company (created in 2015) that provides an internet service designed to unite a variety of different cloud storages in a single interface. The main concept is to make cloud storage simpler for the common user adding the ability to easily share files across different cloud providers and with friends.

Flopfile has chosen to use LSM as their main theoretical foundation since they believe it can provide the best chance for creating a sustainable business. The company has acknowledged they have yet to learn more about the market as well as the theory behind LSM and have therefore agreed to participate in this thesis as a case study.

Since the company is very dependent on their specific software product, they needed help optimizing their development activities. Specifically they wanted to know if they could work smarter to minimize the risk of discarding too much code whenever doing an iteration and getting customer feedback. We have therefore used Flopfile as a test company for evaluating a set of specific technical guidelines that aims to solve the technical challenges associated with LSM.

2) Literature review

In order to suggest guidelines that will help prevent technical challenges in the LSM we first need to gain a comprehensive overview of the literature surrounding LSM. More specifically, we will structure the literature review in the following way:

1. **The evolution of entrepreneurship:** How the LSM differentiates from traditional methods.
2. **Lean Startup Methodology:** An in-depth look at the LSM and its core principles.
3. **Technical challenges:** Barriers and limitation in the software development process when applying LSM.
4. **Technical methods:** Based on these technical challenges we will explore alternative methods used in similar environments in order to create preventive guidelines.

2.1 The evolution of entrepreneurship

Starting a successful business is never an easy task. Evidence show that whether it is a new IT startup or even just a new initiative within an already established organization the results are often hit- or-miss with emphasis on miss (WSJ, 2012). Information technology has advanced at such a high level that almost everything is possible to make, whereas the question switches from "*can it be build*" to "*should it be build*".

The traditional product development cycle which is still being used by many companies today starts with identifying an opportunity, then writing a specification (usually in form of a business plan), pitch it to potential inventors, build the product and finally start selling it (Furr & Ahlstrom, 2011).

Cooper (1986) proposed a new and supposedly more effective approach to the product development process. In his book "*Winning at New Products*" from 1986 he created a blueprint for managing the whole process called the Stage-Gate model. The study used empirical data from more than 60 companies doing new product launches. It describes the process for entrepreneurs or managers going from an idea to launching a product broken down in different stages. Every stage consists of different tasks that must be completed prior to the next stage. If you want to advance to the next stage, you must go through a '*gate*', which usually correspond to a meeting within the company discussing the following points:

1. Quality control
2. Must-meet criteria and should-meet criteria
3. Action plan for the next phase.

The model (Figure 1) consists of seven stages with five corresponding gates after each stage. The purpose of the model is to accelerate the product development and increase the chance of success by taking a very complex structure and breaking it down into smaller and more manageable pieces. We will provide a small summary of each stage in order to grasp the concept of the model.

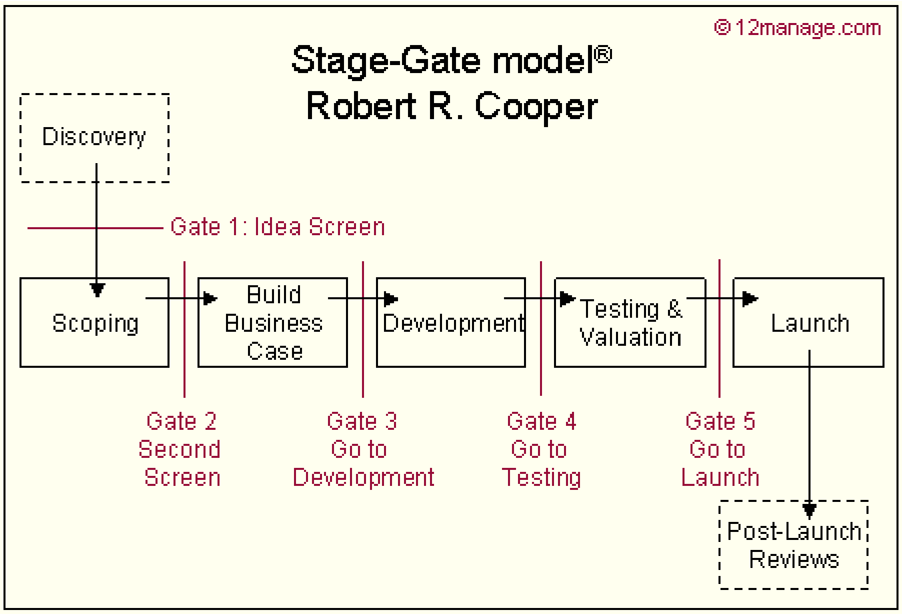


Figure 1 - State-Gate model by Robert R. Cooper (Cooper, 1986)

1. Discovery – Like all innovative and creative initiatives, it all starts with the discovery phase. This is the very early stage in the process where you discover an idea you think is sustainable in the market.
2. Scoping – Exploratory research on the project and limiting the scope and goal of the project.
3. Build business case – A very detailed and in depth business case of the competitive landscape and surrounding environment. The business case should always include a product definition, a product justification and a project plan.
4. Development – The design and development of the product with sufficient product test.

5. Testing and validation – Different kinds of product test, preferable with both real customers and experts in the field.
6. Launch – Market launch with various marketing and sales activities.
7. Post-Launch review – Reviews are performed post launch to gain quality assurance and see how sustainable the product is.

The Stage-Gate model has been widely adopted by various companies for many years by both new startups and established enterprises trying to launch a new product. It has received praise for providing an overview, which enabled easier prioritization and strong compatibility with performance metrics such as IRR, NPV and more. However, recently as the whole Agile methodology has emerged, sequential methods like the Stage-Gate model has received a lot of criticism for being too out of place with the actual market and sometimes even working against innovation (Furr & Ahlstrom, 2011). Furr & Ahlstrom (2011) argues that this type of waterfall model can be ideal for already established enterprises but it is less fitting for new ventures. With new and innovative projects, a high level of uncertainty often characterizes the surrounding environment, which leaves many unknown factors.

The big problem with sequential methods like the Stage-Gate Model is therefore the lack of user involvement and adjustment early on to compensate for this uncertainty. Viewing an entrepreneurship as just another business, leads to an improper use of existing theory on management and organization due to the big difference between large enterprises and new startup ventures (Furr & Ahlstrom, 2011).

The idea of treating a new startup venture as a completely separate kind of entity is one that is shared amongst a lot of the recent entrepreneur-related literature, such as the Lean Startup methodology.

In order to fully comprehend the LSM and why there was such a big need for new and revolutionary techniques we have selected some of the main processes in a traditional product development cycle and looked at the challenges when dealing with startups according to more recent literature.

2.1.1 Business plan

Writing a proper business plan as an entrepreneur has traditionally been considered one of the most important parts in the initial startup process and have been wildly discussed by many well-known authors (Hills, 1998). Up until recently, a business plan has been recognized as a core feature in every startup. However, as new and more agile methods have emerged, the validity of a long-term business plan in a new startup has been questioned.

In fact, very little empirical research exists claiming that a business plan can increase the chances of success in a startup (Gruber, 2010). The discussion on business plans in startup is thus twofold: Some researchers do believe that a business plan is an essential tool in the decision-making process and therefore a necessity in every startup venture (Shane & Delmar 2004). Other researchers claims that the use of prototype and rapid iteration is much more valuable and thinks that traditional business planning in a startup is a waste of time (Bhide 2000).

Steve Blank (2006) is known as being highly critical of using a traditional business plan in a new startup. Like Bhide (2000), he thinks that writing a comprehensive business plan in the initial stage is a form of waste. He presents the argument that since a business plan is based on a set of assumptions, which obviously have yet to be proven; following a long-term plan that is based on these assumptions is far too risky. Eric Ries, who is also an advocate for a new form of business planning argues that creating a long-term plan in an establish enterprise is extremely different from a new venture. Most of the assumptions in these established enterprises are drawn from past industry experience and previous knowledge about the market, whereas in a new startup these are more or less based on some kind of hunch with no empirical evidence. The absence of a long-term business plan can therefore be financially sensible since the entrepreneurs often have very limited resources for initial research about the market (Bhide, 1999). McGrath and MacMillan (1995) argues that traditional planning can even be counterproductive and prevent learning since you are too focused on meeting your initial plan and forget to validate and adjust things in the process.

Even though many authors are rejecting the notion of a long-term business plan in a new startup, authors like Brinckman, Grichnik & Kapsa (2011) have criticized the absence of a business plan and think that a more systematic and prediction-oriented approach is the still the most optimal way when starting a new business. Tim Berry, a recognized entrepreneur and the founder of Bplan.com responded to Steve Blank criticism with the following:

[On Steve Blank's criticism on a business plan] "*That's sort of like saying getting regular exercise isn't good for you because some people overdo it and end up with joint damage.*" (Berry, 2012)

Based on the criticism a new concept called a Lean business plan has been popularized - this recognizes the fact that traditional planning techniques are not suitable for startup ventures but a total lack of planning can also have a negative impact on the project. Instead, the entrepreneurs should constantly validate their assumptions and adjust if needed. This is a key point in the whole LSM, which we will elaborate on later in this literature review.

2.1.2 Decision making

The discussion on decision making in very uncertain environments has received much attention in recent years. David A. Harper (1999) was one of the first researchers to suggest another model for decision-making designed specifically for entrepreneurial ventures, which he calls "*The process of entrepreneurial learning*". This is a scientific process designed to help entrepreneurs test different hypotheses about their product and the marketplace and make informed decisions based on these learnings. Harper draws inspiration from Karl Popper and his theory on the growth of science (and knowledge) which advocates falsifiability, testability and testing. In his book (Harper, 1999), he uses Popper's reasoning cycle and tries to apply it to entrepreneurship in general.

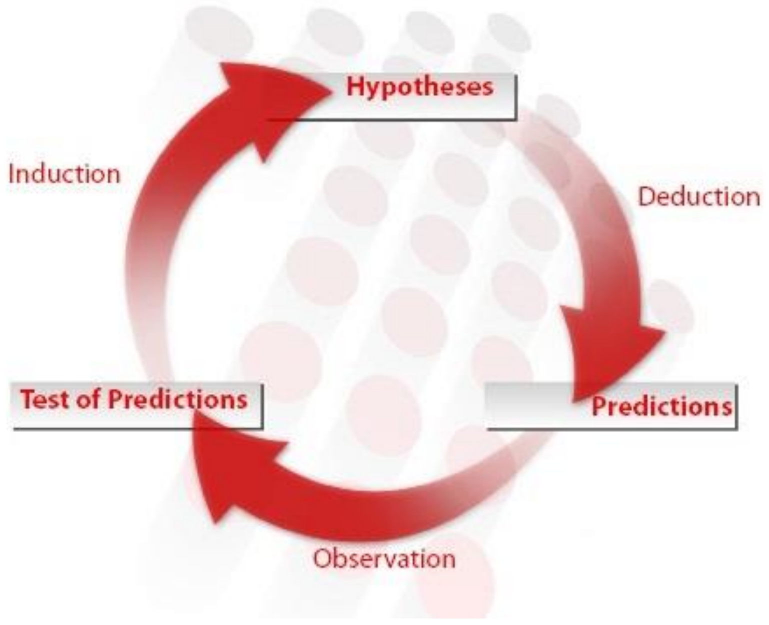


Figure 2 - Karl Popper's model on falsifiability (Shuttleworth, 2008)

In Harper's implementation, the process starts when the entrepreneur is encountering a problem. This problem could be anything regarding their new product or service, e.g. people do not want to create an account on our website. Then, based on the problem you form hypotheses that will solve the problem e.g. a Facebook signup feature will make it easier to sign up and therefore solve the problem. Lastly, you test your hypotheses in the market place and choose to either revise or validate your problem. The new problem will then form a new set of hypotheses that are tested. Even if the hypotheses happens to be true, new problems will arise in the process thus creating this iterative loop of constantly validating your choices. The process allows entrepreneurs to learn from their mistake and discover these mistakes early in the process (Harper, 1999).

Steven Blank used the same basic methodology in his book "*The four steps to the Epiphany*" (2006). The book suggested a new and controversial approach for entrepreneurs, which created the foundation to what would later be known as the Lean Startup Methodology (LSM).

Blank argued that customer development was just as important as product development and these activities should be done in parallel. Right from the early stages, you should always focus on the customer and remember to validate if what you think about the customer is actually true.

A case study done by Silberzahn & Midler (2007) discovered that companies who did product and market development simultaneously were more likely to adapt to changing environment quicker and in general just be more flexible than companies using more traditional approaches.

Based on the findings a completely new methodology emerged which supposedly increased the odds of success in a startup working in an uncertain and risky environment. By combining much of the recent literature on entrepreneurship, the LSM tried to distance itself from traditional methods often used in established enterprises and instead suggested a new and different way of decision-making tailored to entrepreneurs. The new perspective should supposedly speed up the process while at the same time sustaining lower failure-rates. We will now dive into LSM and elaborate on the specific elements within this methodology.

2.2 Lean Startup Methodology

This section will focus on the theory behind LSM in order to provide a solid foundation for the thesis and our research question. More specific, it will address how LSM is structured and how it actually works in practice.

Steve Blank and Eric Ries are the two most acknowledged representatives for LSM and have publish numerous books and articles such as *"The Four Steps to the Epiphany"* (Blank, 2006) and *"The Lean Startup"* (Eric Ries, 2011) about the methodology. Other respected spokesmen include Nathan Furr and Paul Ahlstrom. In 2011, they published a book called “*Nail It Then Scale It*” (Furr & Ahlstrom, 2011) where they provide hands-on tips for entrepreneurs.

LSM is a new way to create and manage new projects specifically designed for new ventures. LSM was officially introduced to the public in 2006 by the pioneer Steve Blank as a new form of management for projects operating in markets under extreme uncertainty. LSM gets its roots from Lean Manufacturing invented in Japan in the 1960's. Lean Manufacturing was created by Toyota Production System to accommodate the customers needs and improve the overall customer value. Toyota wanted to eliminate waste in all areas of their production thereby being able to shorten delivery time, lower the cost and improve the quality. Today, Toyota is the world's largest car manufacturer (Statista, 2014), which is the reason why many of their initiatives are being thoroughly investigated for future learning purposes.

The creation of LSM

Steve Blank created the concept of Customer Development which quickly attracted attention in Silicon Valley. Blank wanted to tell people that the notion of "*Build it and they will come*" is not true for most products - you need to make sure that people want your product before you start developing it. He describes that many startups fail from lack of customers and not from failure in product development. Therefore, it makes no sense not to have processes for managing customer development.

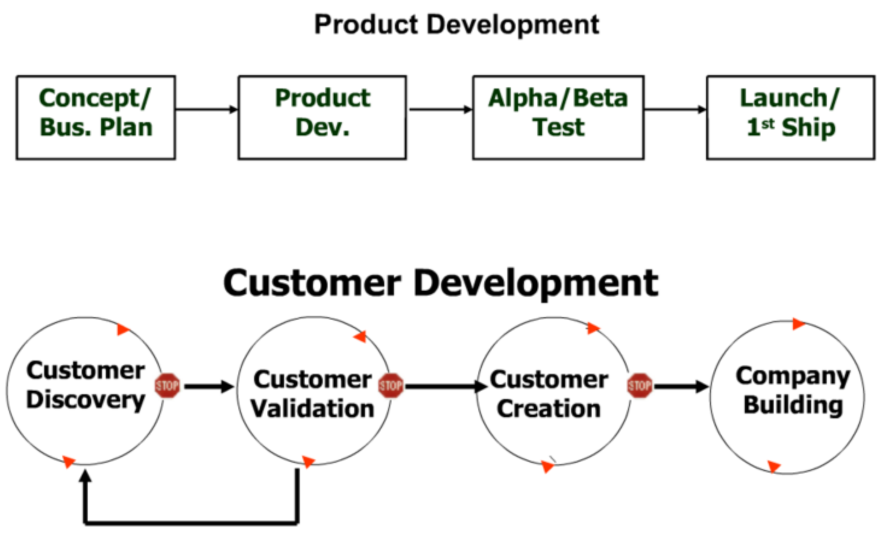


Figure 3 - Steve Blank's model on Customer development (Jstanto, 2008)

Blank created the concept of customer development (Figure 3) as a parallel process to the product development with clear defined and measurable checkpoints. This model is a core feature in the LSM and therefore we will elaborate it later in this section.

After Steve Blank introduced LSM in his book, it has been debated whether or not these principles actually work in practice.

In 2011 one of Blanks former students, a man named Eric Ries published his book *'The Lean Startup'* describing the LSM concepts combined with his own observations during his career as a computer programmer. This book is currently the number 1 bestseller on Amazon within the category New Business Enterprises (Amazon, 2015) as mentioned in the introduction. The reason why Eric Ries has gotten so much attention for his book is partly because it presents the LSM principles in a very tangible way filled with real life examples showing that LSM actually works in practice. Especially one example gives him and his book credibility - his own company IMVU. IMVU, which stands for "*Instant Messaging Virtual Universe*", was founded in 2004 by Eric Ries and three other co-founders and is an online space for meeting new people. In his book, he describes their numerous failures and how they used LSM to get back on track. In 2011, the company reached an annual revenue of more than 50 million dollars with over 100 employees.

Eric Ries explains that his mission with The Lean Startup is to improve the success rate of new ventures. He is trying to achieve this by presenting The Lean Startup Methodology as a journey structured in three categories. These categories are as follows:

1. Vision
2. Steer
3. Accelerate

In this section, we will dive deeper into the first two of the three categories and discuss the different authors view on the LSM principles. The reason why we will not elaborate on "Accelerate" is because it is not directly related to our research question and therefore considered out of scope.

2.2.1 Vision (Getting started)

The Lean Startup Methodology starts by establishing what an entrepreneur is. Ries argues that an entrepreneur is everyone who works with innovation whether it is in a company of 2 or 200. Managers working with innovation in large corporations are sometimes referred to as *'intrepreneurs'* because their mission is to create a startup within the company. This means that the LSM principles and guidelines works with every form of innovation and not just when starting a new company. The LSM is not focusing on how the company can make money but more about how to adapt a specific product to fit the target group and thereby create the opportunity to make money.

One of the LSM principles is that entrepreneurship is management. As Ries describes, a startup is an institution, not just a product. Therefore, the need for new management methods to account for extreme uncertainty is immense. Like mentioned in the business plan section, one should not have a very in depth plan because of this uncertainty. However, a completely “*Just-do-it*” attitude is also not the optimal way.

With this in mind, the entrepreneur is ready to define or specify the grant company's vision.

Creating the Initial hypothesis

Every company starts with a vision for what it wants to achieve with a specific product or service. This vision is often based on the founder's ideas and thoughts. It is often so high level that it might be difficult to derive concrete goals and plans from it. Therefore, LSM starts by breaking down the grant vision into smaller component parts. By doing so, it is easier to specify goals for the vision and creating the possibility to test it.

Testing is one of the core values of LSM, which can be seen in nearly every aspect of its initiatives and guidelines including the initial hypothesis. Often entrepreneurs have ideas they believe potential customers want, but fail to test whether or not this is actually the case. As Blank (2006) describes "[…] *a startup's initial vision is really just a series of untested hypotheses*". If the initial vision is not testing in a startup, the risk of failing increases dramatically (Ries, 2011).

The initial hypothesis consist of two important assumptions; the value hypothesis and the growth hypothesis (Ries, 2011). These are characterized as *'leap of faith assumptions'*, which the whole foundation of the business model relies on.
The value hypothesis is the company's assumption on what the customer finds valuable. This is what the company intends to create hoping that the target customers can see the value in it. Often this is connected to a specific customer problem/pain.
The growth hypothesis is the assumption on how the startup will attract customers. This includes how to gain traction from the early adopters as well as how to repeatedly create an increase in customers.

Blank (2006) and Furr & Ahlstrom (2011) argues that further breakdown of the hypothesis is needed to completely verify the vision. Blank (2006) suggest splitting the initial hypothesis into nine hypotheses according to each of the different areas in the Business Model Canvas. Furr & Ahlstrom (2011) are somewhat in between Blank and Ries. They suggest creating two hypotheses, which should contain a range of different parameters.

What all the authors have in common regarding LSM is their emphasis towards validating the initial hypothesis to be able to make decisions based on facts rather than assumptions (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011).

Validating the hypothesis

In order to validate the initial hypothesis it is crucial for every startup to get feedback from potential customers within the target group. More specifically, LSM addresses that the entrepreneur has to "*get out of the building*" to truly understand the customers (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011).

However, before rushing out of the building the entrepreneur need to plan who should validate the hypothesis. According to Blank (2006) it is important to create a list of customers that are smart, respected and first movers within the target group. This list can become very useful later on for the feedback these visionaries can provide regarding new ideas, flaws and general advice (Blank, 2006). Blank (2006) describes these visionaries as "*The most important customers you'll ever know*" and

calls them *'earlyvangelists'*.
These visionaries can be found in any given way possible to the entrepreneur. When contacting these customers it is important to remember to keep statistic on the hit rate of whether or not the potential customer agrees to an interview (Blank, 2006; Furr & Ahlstrom, 2011). Blank (2006) suggest that if less than 50 percent is interested the entrepreneur has to revise the hypothesis due to low interest in the defined problem or product.

Once the entrepreneur has found interested customers the next LSM principle of validated learning becomes relevant.

Validated learning: "*The process of demonstrating empirically that a team has discovered valuable truth about a startups present and future business prospects. It is more concrete, more accurate and faster than market forecasting or classical business planning.*" - Eric Ries (Ries, 2011)

To discover valuable truth about the target group it is important not to draw conclusions from single customers and be as objective as possible when interviewing (Furr & Ahlstrom, 2011). The entrepreneur should not try to sell the product but instead try to discover what the customer might want to spend money on simultaneously while testing the hypothesis (Blank, 2006; Furr & Ahlstrom, 2011).

When the entrepreneur has gathered sufficient data in order to validate the initial hypothesis, it is time to start developing the solution/product (Furr & Ahlstrom, 2011).

Minimum Viable Product

When developing a consumer product it is important to know what customers want to pay for. One thing is to determine that the customer has a problem and thinks the entrepreneur's idea sounds promising, another thing is to create a solution solving the actual customers problem in a way the customer think is valuable. Ries (2011) describes this as the most difficult part of validated learning. Often customers do not know what they want. Therefore, it might be difficult for them to answer hypothetical questions about a non-existing product.

LSM has created a principle for dealing with this situation called "*Minimum Viable Product*".

According to Ries (2011), a minimum viable product (MVP) is a version of the product designed with the least amount of effort that allows the entrepreneur to test a given hypothesis with the largest amount of validated learning as the outcome. The goal of a MVP is to learn what the customers really want as quickly as possible by testing your leap of faith assumptions or fundamental business hypotheses.

Ries (2011) claims that it is easier to collect reliable data of what customers think when observing them rather than interviewing them. This might also raise questions the entrepreneur would never have thought of. Therefore, in order to create the best opportunity to learn, the entrepreneur needs to build the optimal prototype for this - a minimum viable product.

When creating a MVP with the least amount of effort that still fulfill customer needs, it is not possible to include the full set of features or a final design (Ries, 2011; Furr & Ahlstrom, 2011). Therefore, according to LSM the entrepreneur should create a minimum feature set hypothesis. This hypothesis should include the features the entrepreneur thinks are the most important, based on previous research. This hypothesis should then be validated by contacting customers within the target segment (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011). The LSM states the importance of finding common features for most of the customers within the target group for best results.

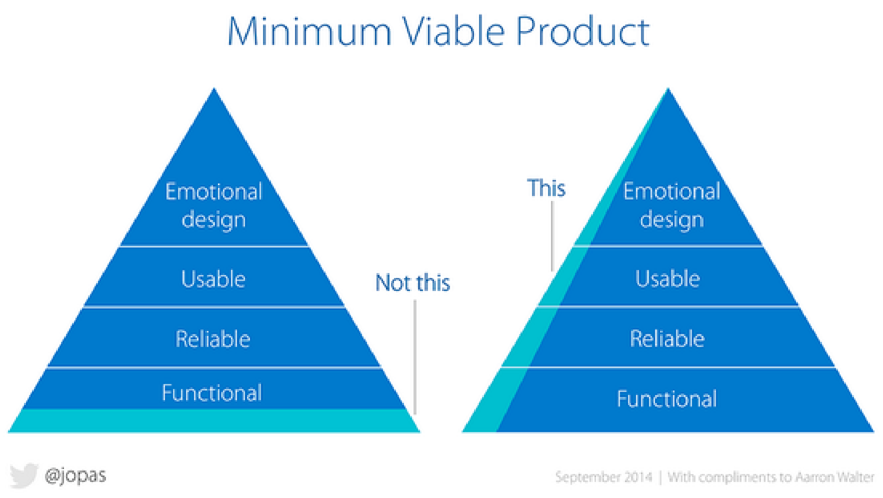


Figure 4 - An illustration on how to create a Minimum Viable Product (Pasanen, 2014)

When a minimum feature set is validated, it is time to build the MVP. This does not necessarily have to be a physical prototype[1] but can just as well be a virtual prototype[2] (Blank 2006; Ries 2011; Furr & Ahlstrom, 2011). Ries (2011) describes how a virtual prototype in some cases can benefit the startup

[1] A physical prototype is a specific version of the product containing some of the final products features (Eg. a Website with limited functionality).
[2] A virtual prototype is a presentation of the actual product (Eg. a video, a text description, a picture etc.)

when a physical prototype might take too much time and effort to create. He gives an example with Dropbox who created a video, presenting the product and how it could help the customers. This increased the number of users signing up for the beta waiting list from 5.000 to 75.000 overnight.

If the entrepreneur choose to make a physical prototype, LSM provides some guidelines for doing this properly. One thing is that the entrepreneur should not focus on high quality as an important task. This is because the high quality seen from the entrepreneur’s perspective might not be seen the same way from the customer’s perspective and vice versa. The goal of a MVP is to transform the feature set into an actual product to be able to learn from customers, therefore if low quality is an issue this is the perfect opportunity to find out (Furr & Ahlstrom, 2011). Another guideline is to remove every feature that does not contribute directly to the learning the entrepreneur seeks in order to get more accurate data when testing (Ries, 2011).

Although LSM advices startups to build a MVP there are some potential risks included (Ries, 2011). First, if the products needs patent protection, releasing a MVP might narrow the time for applying for a patent. Second, there is the risk of a large corporation stealing the idea. Third, there is the risk of damaging the company name by releasing a low quality product. However, Ries (2011) denotes these risks as being rather small since large corporations do not have the time to look into every startup and the company can just release a MVP under a different name.

2.2.2 Steer (Justified decision making)

After having built the startup’s first MVP the entrepreneur has created the foundation for future growth. The next step is to improve this so in the end the entrepreneur can have a successful company. To do this Blank (2006) describes two ways to steer the development process of the product - "*Faith-based*" and "*Fact-based*" decision making.

The Faith-based approach is the easiest way for an entrepreneur to steer the startup although also the most dangerous one (Blank, 2006; Ries, 2011). When using the Faith-based approach the entrepreneur creates hypotheses on what the customers want and which customers might be interested but instead of testing these hypotheses with actual customers, the entrepreneur tries to answer them on his own. Thereby he uses gut feeling to build the company.

According to the LSM, the Faith-based solution creates too high risk of failing from lack of customers. Instead, one of the core values of LSM is to use Fact-based decision-making on every aspect of the startup to be able to validate whether or not the customers are actually interested in the company’s specific product (Blank, 2006; Ries, 2011). The LSM advises that you start testing the most critical and riskiest assumptions that are the foundation for the business model. If these assumptions are proven inaccurate, the later they are discovered the more time and money might have been wasted (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011).

Test

To be able to validate a startup's hypotheses (during the products development phase) the entrepreneur will have to test them with actual customers. In order to do this Blank (2006) has created a model called Customer Development where he validates every aspect of the Business Model Canvas. The customer development model is divided into four iterative steps. This means that every step is repeated several times in order to adapt to changes once the hypotheses have been validated.

The four steps are (as seen in Figure 5) "*Customer Discovery*", "*Customer Validation*", "*Customer Creation*" and "*Company Building*". This model enables the company to adapt more easily to changes than prior management models.

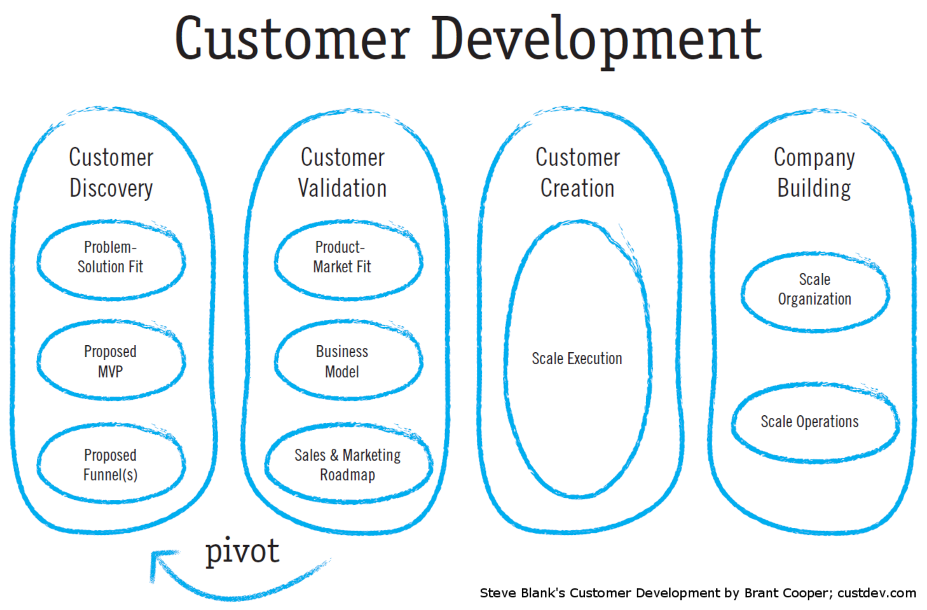


Figure 5 - Steve Blank's model on Customer Development (Cooper, 2010)

Blank (2006) starts with the step called Customer Discovery. This is where three specific hypotheses are tested; which customers are you going to sell to? Does the customers think the problem is relevant? Does the solution fit the customer?

When the entrepreneur has received sufficient learning outcome from testing these three hypotheses, it is time to proceed to the next step, which is Customer Validation. Here the entrepreneur should start creating sales material and try to sell the product to early customers. Even though customers have said the solution, fit their needs it does not necessarily mean they would want to buy it. The entrepreneur should then measure if the output of these sales tests are as predicted. If things are not going as planned it is time to pivot and take a step back in the customer development model. Otherwise it is time to persevere/continue with the next step (Blank, 2006). Furr & Ahlstrom (2011) agrees with Blank and emphasizes the importance of what they call breakthrough questions and price points. Breakthrough questions are tough questions for the entrepreneur that has a high importance like if the customer would pay money for the proposed solution or not. If the customer answers positively it is important to discover how much the customer is actually willing to pay (price points).

These two steps are the most important ones when it comes to steering the development process in the right direction (Blank, 2006). The next two steps (Customer Creation and Company Building) are focusing on how a company can grow and is out of scope for this thesis.

As a critique to Blank's Customer Development model and his book in general, Ries (2008) argues that Blank tries to do too many things at once. He states the following about Blank's book:

"*This is a self-published book, originally designed as a companion to Steve's class at Berkeley's Haas school of business. And Steve is the first to admit that it's a "turgid" read, without a great deal of narrative flow. It's part workbook, part war story compendium, part theoretical treatise, and part manifesto. It's trying to do way too many things at once.*" - Eric Ries (2008)

Instead, Ries (2011) proposes to test the different hypotheses using what he describes as the "*Build-Measure-Learn*" feedback loop. Ries (2011) has created this model in order to optimize the time spent from defining a hypothesis until it has been validated. The general principle of this model is structured around the LSM principle of how you should iterate rapidly (Ries, 2011).

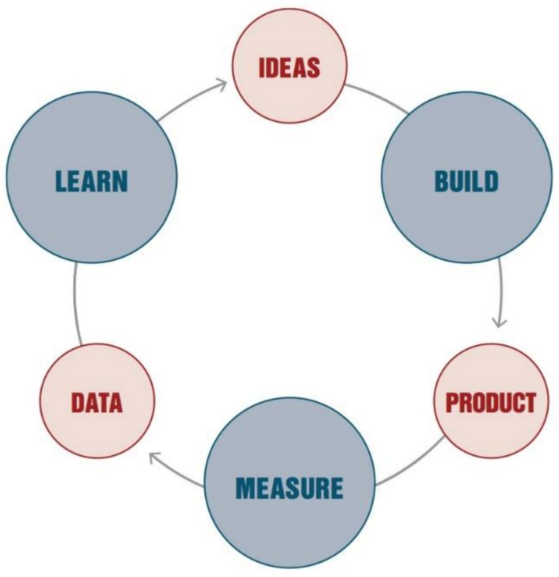


Figure 6 - The "Build-Measure-Learn" feedback loop (Ries, 2011)

The Build-Measure-Learn feedback loop is illustrated in Figure 6. As shown with the lines and arrows the six circles/steps are connected two by two and is often just referred to as the build, measure, learn steps. Each step has to be completed in order to proceed to the next. The six steps includes the following elements:

1. **Ideas**: The entrepreneur has some ideas for a product/feature/problem he thinks might be relevant to the customer. Therefore, he creates hypotheses to test his assumptions.
2. **Build**: A Minimum Viable Product (MVP) is built based on the previous step in order observe the customers reaction to a specific product.
3. **Product**: Once a MVP has been created, it is time to find specific customers within the target group.
4. **Measure**: A set of measurements and goals are defined for this specific test in order to determine whether the test has been successful or not. The test is carried out and the results are collected.
5. **Data**: The data from the tests are then structured and overall conclusions are derived from them.
6. **Learn**: Once the conclusions have been made it is time to reflect about what has been learned. In this step the entrepreneur then has decide if it is time to pivot (make a sharp change in direction) or persevere (continue with minor adjustments).

Ries (2011) describes the Build-Measure-Learn feedback loop as a steering wheel designed to make constant adjustments on the way to success instead of making complex plans that are based on numerous assumptions. This way the entrepreneur is able to waste less time and money on the project as a whole.

Furr & Ahlstrom (2011) does not suggest a specific model for testing different hypotheses but in general follow the same structure as Blank (2006) and Ries (2011), by iterating through the process. Although one thing is worth noticing, Furr & Ahlstrom suggest that you should always start with a virtual prototype (as described in MVP) in order to minimize the time spent per each iteration. More specifically, they suggest that when the virtual prototype is tested with potential customers an interview guide should be created. This guide is a description about the customer's pain regarding the problem, how the pain is being solved today and what the customer thinks about the proposed solution. Furr & Ahlstrom (2011) states the importance of not making the interview questions too complex so the customer has to create the solution neither too simple so a yes or a no is sufficient.

Measure

All the authors of LSM advices to iterate rapidly and often during the development, but how does the entrepreneur determine if the project is improving? Ries (2011) states that in order to measure progress the entrepreneur needs to use accounting. According to Ries (2011), accounting can be used to set up milestones and follow up. Although general accounting faces a problem. According to Ries (2011), startups are too unpredictable to forecast with regards to revenue. Therefore, Ries (2011) suggest using another form of accounting - The LSM principle called "*Innovation Accounting*".

Innovation accounting is a method that uses general accounting elements and techniques but instead of only looking at future revenue, it distinguishes itself by focusing on different metrics (e.g. Increase in webpage visitors etc.). Ries (2011) states the importance of selecting different metrics for different tests depending on the specific goal of the hypothesis. Selecting bad metrics that does not represent the hypothesis can end up providing bad test results leading to wrong conclusions. Ries (2011) calls these metrics "*Vanity metrics*". Vanity metrics are often positive metrics, but does not tell if a specific product improvement is the cause for this.

Therefore when selecting specific metrics the entrepreneur should look for three characteristics (The three A's); Actionable, there should be a clear cause and effect, Accessible, the results should be simple to understand and access, and Auditable, the results should be credible when read by others (Ries, 2011).

Ries (2011) describes how many startups tend to discuss what would improve their product, implement several changes at once and then celebrate if there is any positive increase in any metrics. Instead, he suggest to use a four step approach when testing:

1. Set clear baseline metrics
2. Create hypothesis to improve these metrics
3. Create a plan on how to test
4. Test and evaluate changes to the baseline metrics

Using this approach together with "*the three A's*" will minimize the time wasted on false assumptions.

Blank (2006) and Furr & Ahlstrom (2011) does not describe in depth how to measure success. Blank (2006) describes the importance of testing a product until the sales person in the company believes he can sell the current product repeatedly. Furr & Ahlstrom (2011) argues that testing needs to be performed until it perfectly matches the customer's needs.

In general, all the authors agree that the entrepreneur should never base decisions on single customers, but always verify problems using multiple customers before changing anything (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011).

Pivot or Persevere

Based on the empirical findings from the conducted tests, it is time to decide whether to pivot or persevere. According to Ries (2011), the difficult question that many startups face is when to pivot and when to persevere. Ries (2011) describes that many startups rely on the rule of thumb that if you can see an increase in your measurements then you persevere. Ries (2011) argues that this is not always the best course of action. Instead, the entrepreneur should compare the results from the different tests to the startups growth model. If the results does not meet the milestones stated in the growth model the entrepreneur should consider to change strategy and thereby pivot.

As illustrated in Figure 7, a change in the product alone is not classified as a pivot but solely an optimization. However if a change in the product includes a change in the startups strategy and the direction the startup is heading then it is classified as a pivot.

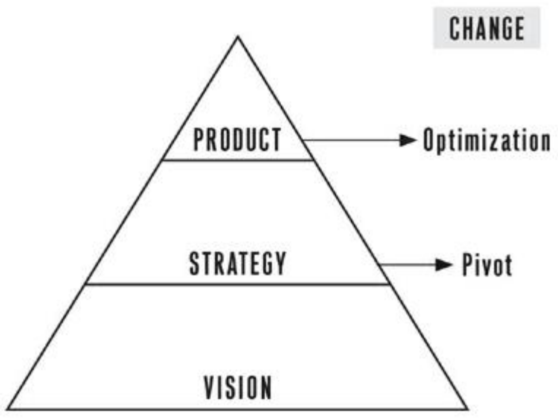


Figure 7 - The Lean Startup Pyramid (Ries, 2011)

In order to better classify what is categorized as a pivot and what is not, Ries (2011) has created a catalog of 10 different types of pivot. Some of these are:

- Customer segment pivot - when a change in the startups customers segment is made,
- Customer need pivot - when a change in which features are most relevant is performed,
- Platform pivot - when a change in platform occurs (eg. from website to smartphone app)
- etc.

It is important to keep these different kinds of pivot in mind when evaluating the results collected from testing the hypotheses. This way the entrepreneur is aware of the importance to act fast and pivot if necessary instead of prolonging the pain (Ries, 2011). As LSM describes it is important to learn as fast as possible. Therefore, the entrepreneur should have a regular pivot or persevere meeting. Not as often as every 2 weeks and more often than 2 months. It is up to the entrepreneur to discover what fits (Ries, 2011).

When the entrepreneur has created a product and has started selling it to early adopters, it is time to focus on how to grow (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011). This includes creating a go-to-market strategy, scaling the customer base and scaling the company. Since the purpose of this thesis is to optimize the development process, this part has been excluded and is considered out of scope.

2.3 Technical challenges

We have now clarified how the Lean Startup Methodology operates in comparison to traditional methods and stated what makes LSM unique. Although the principles of LSM sounds very promising, LSM has also received criticism for neglecting the technical complexity in startups that rely on software development as one of their key activities.

Concerning our research question, we will now dive into specific technical challenges that arises when applying LSM to these technical startups.

2.3.1 Architecture

As previously mentioned in the introduction, LSM has been criticized for devaluing architecture in the software. Michael Sharkey, the CEO of Bislr (a successful email marketing service) wrote an article called *"6 things wrong with the Lean Startup Model",* where he criticized LSM based on his own experiences and knowledge gathered in his company. One of his critiques addresses the architectural challenge when applying the LSM principle "Minimum Viable Product":

"Companies that focus on MVP tend to skimp on architecture [...] if you don't have time to build a whole product you also won't make the time to invest in architecture. Sadly, no decision about architecture is a decision, one that will determine your success or failure as a company."
(Sharkey, 2013)

Absence of a well defined architecture can have very negative consequences. The results can easily be a so called *"spaghetti architecture"* where all the different components in the program are tangled together in one big unstructured mess. This means that if requirements changes and you need to change a single part of the program, other parts that depends on this component stop working (Abbott & Fisher, 2009).

Another consequence affected by badly designed architecture is the scaling potential (Abbott & Fisher, 2009). Abbott & Fisher (2009) argues that poorly designed architecture can completely hinder the scaling possibility of a given application. Sometimes it is discovered rather late in the process that scaling is actually an issue and by that time it might be too late to change. In the worst case, the company will have to completely redo the application.

However, simply adding some architecture into your software will not solve the problem - traditional software architecture is not optimized in extremely uncertain environments with constantly changing requirements, which is the case when applying the LSM principle "Iterate rapidly". (Coplion 2010).

"Classic architecture tends to be fearful of large changes, so it focuses on incremental changes only to existing artifacts: adding a new derived class is not a transformation of form (architecture), but of structure (implementation)" (Coplion, 2010)

It is clear that there does exist a mismatch between applying LSM to your startup and having a well-defined architecture that is optimized to the conditions you face. If your code is not adaptable to changes, it will create unnecessary barriers and challenges every time a pivot is made.

2.3.2 Innovation

Innovation is a key factor in almost every startup. To get a competitive edge entrepreneurs will have to think outside the box. Although this is common sense for most people Cohn (2014) has criticized the software development methodology Scrum in some areas that directly relates to some of the core principles of LSM. He argues that using principles as "Minimum Viable Product" and "Iterate rapidly" from the LSM hinders true innovation. The idea of launching the product as fast as possible can sometimes overshadow the overall quality of the product.

"... many teams have become overly obsessed with being able to say they finished everything they thought they would. This leads those teams to start with the safe approach. Many teams never try any wild ideas that could lead to truly innovative solutions." (Cohn, 2014)

2.3.3 Eliminate waste

Lean is built on the foundation of working as efficiently as possible. Eliminating waste is at the core of LSM and incorporated in every principle.

LSM has been criticized for not aligning well with traditional software development methods (Sharkey, 2013). Things like detailed requirement specification and various task descriptions are key activities in many developer teams. However, the uncertainty of the environment and the constantly changing requirements based on new empirical knowledge can easily render this method obsolete and produce waste.

"The waterfall stereotype is patterned around greenfield development. It doesn't easily accommodate the constraints of any embedded base to which the new software must fit, nor does it explicitly provide for future changes in requirements, nor does it project what happens after the first delivery" (Coplion, 2012 p. 11)

Many companies have instead adapted a more agile approach to their software development process. Although this does have more synergy with LSM it also presents challenges when they are working in tandem. Agile methods are build in order to accommodate changing requirements during the development phase. The problem is that well known Agile methods like Scrum are not properly optimized for startups. Specifically the high degree of uncertainty as we see in startups is a challenge for agile methods. This is partly because the different "Sprints" (a notion from Scrum) are build around fixed time intervals between each release where it has been planned that a set of features should be created. At the same time agile methods does not include the LSM principle "Validated learning" which means that some of the planned features in the sprint might not be in the interest of the customers. This increases the risk of producing waste by scrapping code.

2.4 Technical methods

After having discovered some of the technical challenges arising when applying LSM to technical startups we have analyzed various technical literature in order to identify how these challenges can be prevented according to different researchers. This section will dive into important literature surrounding the creation process of specific theoretical guidelines to overcome these technical challenges.

In order to construct these customized guidelines we will start by defining different approaches for a software development process (SDP). It is important to gain an overview of how different approaches work in order to create synergy between our guidelines, the SDP and the LSM. If the SDP is fitted for the external environment one faces in a startup it can help minimize the amount of waste produced during development.

2.4.1 Approaches to a Software Development Process

This section presents a brief overview of some of the most dominant SDPs and their different attributes. This is important in order to identify how a SDP should be structured in order to fit the extremely uncertain environment in startups.
When developing software, organizations have to adapt their approach to the different elements surrounding the project. These elements are the requirements, the technology and the people involved in the process. The interactions between these elements are called the complexity concept (Schwaber, 2010). According to Schwaber, any software project can be described by how easy it is to understand and how easy it is to predict.

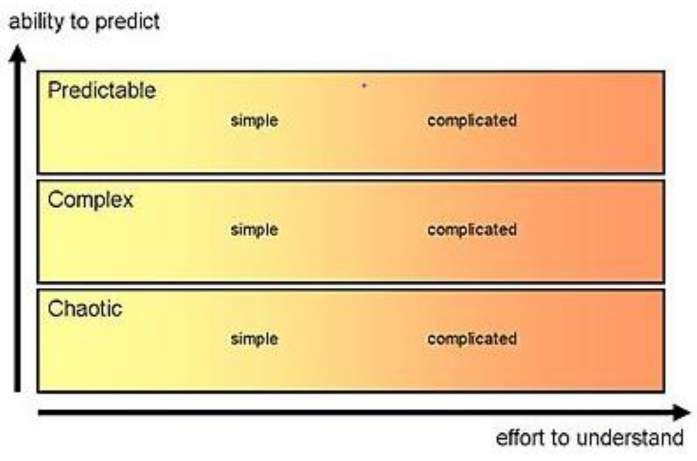


Figure 8 - Breakdown of software project by its predictability and understandability (Appelo, 2008)

Depending on the ability to predict a software project, it will be classified as either chaotic, complex or predicable. By using these defined categories, it is easier to compare different projects, and select appropriate methods. Regardless of its predictability, a system can also be either simple or complicated. The definition of a simple system relates to the technical complexity required to develop it. If the number of components and interactions are limited then it is a simple system since you can easily comprehended the entire system (Appelo 2008). For example, the classic video game Pong would now be categorized as a simple, predictable project since the game itself has a limited amount of components and the rules are straight forward. Games created more recently are usually categorized as complicated and complex since the amount of components and interactions are far greater, even though they are still limited overall.

If you abstract higher and try to look at the entire software development process instead of just the product, the predictability and understandability changes quickly. Schwaber (2010) argues that software development is *always* a complex process but can be either simple or complicated. He says that the only way to understand a complex project is by studying how the whole system operates and not only identifying all the parts, which you would do with a predicable project. A project is chaotic if it is not predicable at all. Even though startups works in highly uncertain environment it will still not be categorized as chaotic since there will also be some kind of predictability according to Schwaber (2010).

Prescriptive and adaptive methodologies

When the complexity of a system has been analyzed, an appropriate methodology for the software development process can be chosen. There are many known methodologies used in software development each with a different number of constraints or rules. The choice of methodology usually depends on the surrounding environment and requirements in the project. Some projects could require a total regulated process where every element can be predicted. Here a very plan-driven approach would be optimal. Others will require a more adaptive methodology where requirements can change and you cannot predict every element. Based on Kniberg & Skarin (2010) findings, Norrmalm (2011) made a comparison between the most known methodologies used in software development.

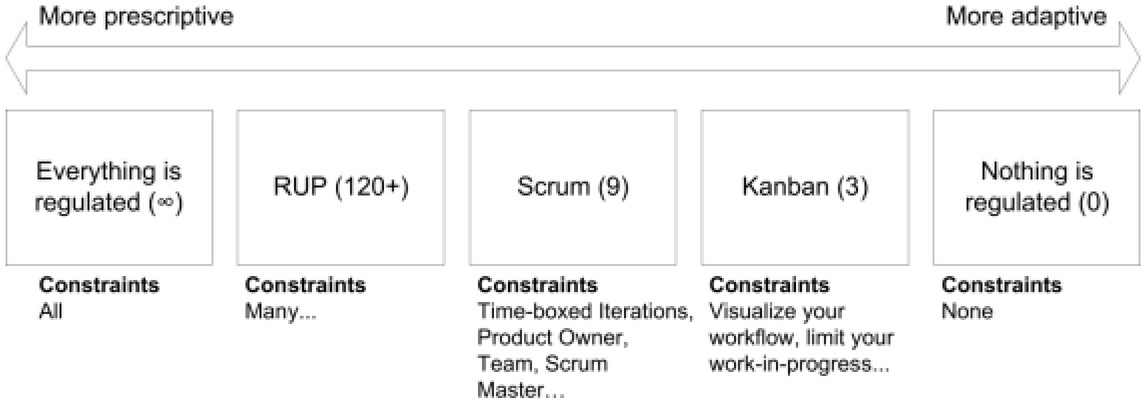


Figure 9 - A comparison of the most known methodologies used in software development (Norrmalm, 2011)

Figure 9 compares some of most used methodologies by how regulated the process is. If everything is regulated the process becomes fully prescriptive and if nothing is regulated it is very adaptive. The constraints you see in the model means that you are free to adapt the methodology as you like if you comply with all of the constraints - the greater the number of constraints, the less adaptive the methodology would be and vice versa.

A prescriptive methodology focuses on planning and describing the plan in as much detail as possible. Every single element has been considered and the outcome has been predetermined. The plan does not leave room for change and therefore suddenly adapting to new requirements can be very difficult for prescriptive teams. Since the project plan is optimized for the initial requirements, even relatively small changes would require a completely new plan, which would be extremely time consuming. This means it is only the most valuable and critical changes that will be considered in these types of projects (Boehm, 2004).

Adaptive methodologies on the other hand are very agile and great for markets with a high degree of uncertainty. An adaptive approach means that you would never go into detail of exactly what you should do but rather focus on the overall features needed to be implemented (the prescriptive level is low). Because of this, it can be hard for adaptive teams to portray precisely what will happen long term (Appelo 2008).

It is obvious that a prescriptive method would be a very bad fit for a new startup since requirements are constantly changing and the software should be able to adapt to this. We will therefore analyze some of the Agile methodologies and discuss what happens when you introduce the concept of Lean into these.

Agile methodologies

More and more companies are embracing Agile development as a viable methodology that delivers value to the customer faster in both Information Technology and across key business units. This means that if you work in an agile fashion you mitigate risk by continuously sharing your work with your customers in order to validate what you are building is what they really want. This relates very much to the LSM thinking but with a stronger focus on the software and the implemented features.

Scrum

Scrum is recognize as one of the best agile development practices in use today. It helps teams deliver value in small iterations of 30 days or less. (Schwaber 2001).

Scrum as a framework does not provide many answers for everything that you should do. Instead, it focuses on many questions for your organization about inefficiencies and team allocation that the people in the organization needs to be aware of. Scrum is thus not a prescriptive process but rather exposes many questions relevant within the SDP.(Schwaber 2001).

Scrum works with the assumption that users generally do not know exactly what they want from the start. Therefore their requirements can easily change in the middle of the project. Scrum handles this by adopting an adaptive approach and accepts that the customers problems and requirements cannot be fully understood immediately.

Scrum does this by developing products in an iterative incremental manner. Each iteration is called a sprint. Each sprint contains the following:

- Requirement definitions
- Product design
- Coding
- Testing

The end result of every sprint is a potentially shippable product. Each sprint after that will improve the product a bit more, building on top of it.

The requirements are kept in a '*product backlog'*, which is an ordered list of the features, bug fixes, non-functional requirements etc. that the product currently has. Requirements in Scrum are called user stories because they are focused on what the customer or user needs, they are written in a canonical form to put a focus on the customer. Once the product backlog is finished, the *Scrum Master* can start a sprint planning meeting where requirement definitions are discussed with the customer or user.

The scrum master is an essential concept in Scrum - it is a person who facilitates the sprint planning meetings and ensures that the project runs smoothly by removing any obstacles that prevents the team from doing their work.
During a sprint, it is not allowed to change the requirements; this can only be done at the end of the sprint to avoid misunderstandings. Each morning a *standup meeting* takes place where it is quickly discussed what each person has done yesterday and what they are planning to do today.

Figure 10 illustrates the Scrum process:

1. The requirements are put in the product backlog in the form of user stories,
2. Features and bugs are chosen from the product backlog to be part of the sprint backlog
3. During a sprint all requirements are locked.
    a. During a sprint there are daily stand up meetings.
4. A new working version of the software has now been created and the cycle begins again.

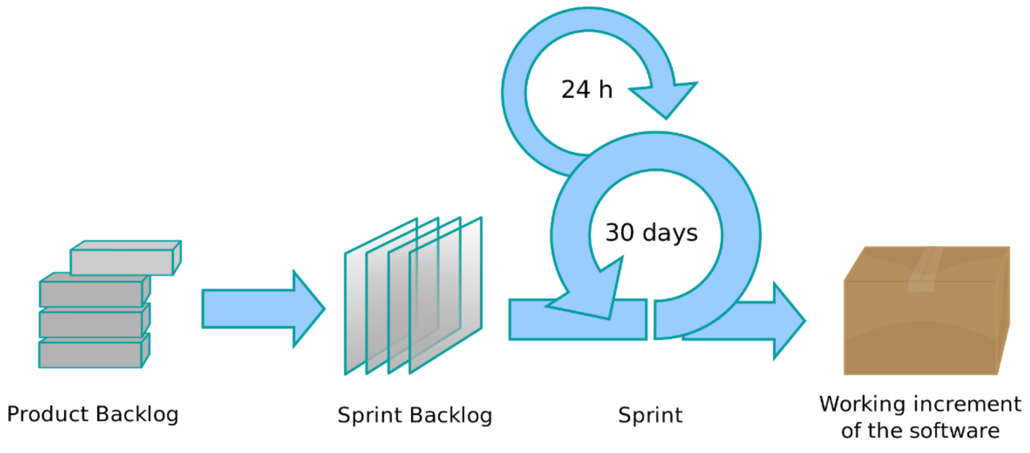


Figure 10 - Illustration of Scrum (Wikipedia (scrum), 2015)

The scrum approach promotes adopted planning, early delivery and continuous improvement instead of doing things in a chronological order and planning long term. Even though Scrum has been widely adopted by many companies, it has received criticism for being too focused on checking the boxes and less focused on innovative solutions (Cohn, 2014). Because of the very small timeframe within a sprint, it can be hard to explore alternative and more risky solutions and instead just go with the safe options. This could worst case render the used technology obsolete over time.

Kanban

Kanban is another Agile framework. It originates from Japan and literally means billboard, which essentially is the purpose of the framework. The basic idea of Kanban is trying to match the amount of work-in-progress elements the teams are currently having, with the resource capacity of the team. It does this by visualizing all the different tasks in the project on a (preferably) physical blackboard. Much like Scrum the different tasks are written down and placed on the Kanban board. However, instead of being constrained by time they can be pulled at any time only limited by the number of work-in-progress tasks allowed at once. This work-in-progress limit is something the team decide for themselves depending on the available resources in the company (Schwaber, 2010).

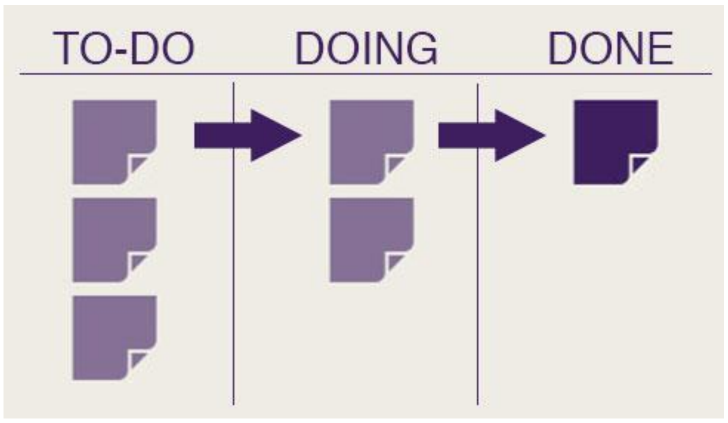


Figure 11 - Illustration of a Kanban board (leankit, 2015)

The Kanban method obviously draws inspiration from Scrum while trying to limit the number of work-in-progress tasks in the pipeline. It also allows teams to measure their lead-time, which is how long it takes on average for a task to be complete.

However, due to the maximum number of *in progress* tasks allowed at once, the team will have to finish some of the current tasks before selecting new once. This means that items stuck in the workflow can choke the process and create unnecessary downtime.

Schawber (2010) has criticized Kanban for the fact that people can be interrupted any time and effectively still works in Silos.

"People are not allowed to work in containers, sharing skills and knowledge to bring complexity into solutions – instead they are worked on a pull (more sophisticated than push) production line."

(Schwaber 2010)

According to Schwaber (2010), people who choose Kanban over Scrum faces substantial risks since they still work in silos and do not share experience and knowledge across the entire team.

2.4.2 Implementing Lean into agile software development

We have now discussed some of the known methods within the agile software development area. These are essential in order to create guidelines designed for software development in very uncertain environments. Another very relevant topic that can strengthen the justificatory knowledge used to shape these guidelines is *'lean software development'*. This is essentially a translation of the lean manufacturing principles into the software domain (Bjørnvig, 2010). It is important to recognize that lean software development does not focus on the Lean Startup Methodology specifically but rather the original concepts of lean derived from the Toyota production system as described in our LSM section.

Lean software development

The most central part of Lean software development is how you eliminate waste. Waste is essentially everything that does not add value to the customer. In software development, this can be unnecessary code, unclear requirements, delays etc. If the result does not strictly depend on a certain activity then it is classified as waste (Coplien, 2010).

Lean software development is often seen as a foundation for Agile methodologies. Scrum inventor Jeff Sutherland describes Lean and Scrum as two separate and complementary developments that both emerged from observing adaptive systems (Coplion, 2010). One central difference is how Scrum is all about '*doing'* while Lean is about '*thinking'* (about continuous process improvements in order to eliminate waste) and then '*doing'*. If we continuously consider each decision, a lot of unnecessary work can be avoided (Coplien 2010). Ballard (2000) argues that small amounts of rework in design can actually add value in form of valuable knowledge.

The goal of lean software development is to create as much value as possible to the customer and in the most efficient manner. In order to accomplish this it require the team to have a commitment to continuous process improvement as we saw in Agile development. It is important to recognize that software development needs humans to solve problems. This means that while the team should still understand the problem, the relevant person must be given authority to come up with the optimal solution.

Lean software development principles

Lean software development is based on seven key principles to help facilitate the lean thinking approach (Poppendieck, 2003). We have selected three of these principles since they are the most relevant in a startup environment.

1. Amplify learning
2. Decide as late as possible
3. See the whole

Amplify learning

"Development is an exercise in discovery, while production is an exercise in reducing variation" (Cobb, 2011 p.25).

In order to amplify learning the team needs to communicate very often. Decisions should always be made based on empirical findings instead of assumptions. The principle takes the notion of validating learning as we saw in the LSM and applies it to software development. In order to successfully develop software features and functions the specific steps used to do this should be repeated throughout the project until the specific customer requirement are met. However, it is also important to recognize that the development team should freely embrace improvements to these steps as they continue performing their specific project tasks (Cobb, 2011).

The team should also be encouraged to expect and embrace mistakes since they will inevitably occur on the project regardless of what agile method is being implemented. The important thing is to learn from these mistakes in order to ensure that the next iteration runs more smoothly and efficiently (Cobb, 2011; Poppendieck 2003).

Decide as late as possible

Another key principles of Lean software development tells us to expect uncertainty and make decisions when uncertainty is at a minimum. By incorporating late decision-making in your development practice it will provide an options-based approach which are effective in uncertain environments. This is because it allows the team members to avoid locking in decisions until you have more empirical data to base your decisions on. According to lean software development, it is more sensible and valuable to delay decisions until they can be made based on facts and not speculation. Keeping design options open in lean software development is more valuable than committing early (Cobb, 2011; Poppendieck 2003).

See the whole

The last principle focuses on seeing the effort of the team as a whole rather than a group of individuals. One of the most difficult problems encountered in software development is that the experts has a tendency to focus on the performance of a specific area of the product. This area usually represents their own specialty and therefore the overall system performance is neglected. In lean software development there needs to a focus on the overall system performance (Cobb, 2011; Poppendieck 2003).

The lean software development principles are a great toolkit when working in a very uncertain environments as you do in startups. It provides an understanding for why many of the agile methods work and offers insights to how you can improve your software development process (Ambler 2010). However, there is one element these principles does not consider – proper use of software architecture. Software architecture is a crucial element in any software process when you are working with complex programs and needs to be able to adapt to big changes in the requirements. For this reason, we will now discuss how you can create a *'lean'* inspired architecture, which takes the environment in a new startup into account.

Software architecture

Developing code takes times, a lot of time. If you have to recreate big parts of the software every time a change happens it quickly becomes a much bigger investment than initially thought, especially in startups with very limited time and resources. Therefore, we need some artifact to find a way around this. One of the best way to do so is with a comprehensive architecture. By combining the Lean and Agile methodology, we can create an architecture tailored to situations where requirements are extremely uncertain and thus are changing constantly (Coplion 2010).

The term *architecture* first came into the software domain in the 1960 where Fred Brooks, a software engineer for IBM saw a tight correlation between traditional architecture and software engineering (Coplion 2010). Architecture in software development is about how you communicate the overall form and avoid clutter in your code.

In order to discuss how an architecture tailored for the LSM can be developed it is important to understand the concept of a programming architecture and what it is trying to accomplish within software development.

A programming architecture simply describes how you structure the code. In the programming world, there are many different architectures which all vary slightly. However, they are all trying to accomplish one common goal; separation of the different concerns in the program. This section will be based on the most popular architectural framework; the Model-View-Controller (MVC) pattern.

Model-View-Controller (MVC)

MVC has been widely adopted as an architecture both in desktop and web applications. The architecture was designed in 1978 by Trygve MH Reenskaug (Reenskaug, 2003) to assist with contemporary issues with the programming language Smalltalk. MVC architecture was invented to overcome tight coupling between the user interface and the back-end code. Deacon (1995) describes how it is normally seen that applications change the user interface overtime, but keeps most of the backend code. He gives the example of the banking industry:

"A banking application that used to sit behind character-based menu systems or command-line interfaces is likely to be the exact same application that today is probably sitting behind a graphical user interface (GUI)." (Deacon, 1995)

MVC is divided into three categories: Model, View and Controller. These three categories each have their own responsibility when it comes to dealing with different elements within the system.

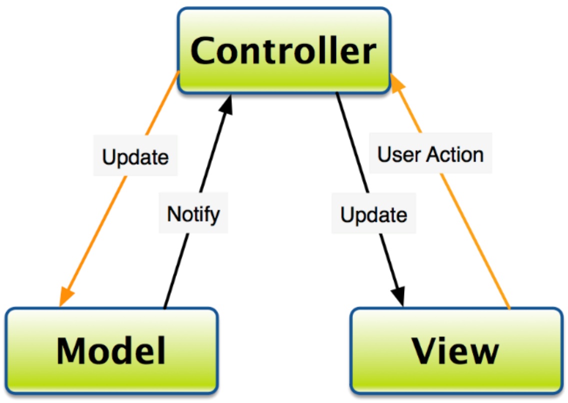


Figure 12 - Graphical representation of MVC (Wikipedia (MVC), 2015)

Model

Model defines the structures of the individual data objects the program contains. It stores and extracts data from the database when receiving a request from the controller. The model is always constant and never changes during the execution of the application. This means that the structure itself never changes, but the specific data can still be extracted. Model should only know itself and never try to manipulate data (Deacon 1995).

View

View contains the user interface, which is what you see on the screen. Views should simply just display data and transmit user input to the controller. E.g. if a user pushes a button it should send a request to the controller that the button has now been pressed by the user.

Controller

The controller handles all inputs and logic in the application. It notifies the model that specific data should be changed and updates the views accordingly (Burbeck, 1992).

Typically, the architectural pattern works like this. A user visits your website, this will send a request to the controller, which will fetch data from the model, manipulate this data if necessary and finally update the view to display the correct information. This ensures that the three components are separated and although there will always be some kind of dependency, the goal is to minimize it. The idea is that, if you want to create a completely new interface this will not interfere with the rest of the program, you simply swap out the view component for another and the program still works as intended.

Lean Architecture

The MVC pattern is a classic software architecture that easily separates the important components in the program, which makes the program more robust and scalable. However, MVC is not optimized for highly uncertain environments. Therefore, we will analyze which components are needed in order to create a well suited architecture for startups.

According to Coplion (2010) software architecture in uncertain environments should focus on the associations and attributes of the program and not on the concrete methods and data members. He describes this kind of architecture as a *'lean architechture'.*

"If we capture the form without expanding into full structure, we stay Lean. That leads to an architecture that can scale and evolve better than one cluttered with the structure of premature implementation." (Coplion 2010, p.81)

Lean architecture focuses on how we develop the overall system based on experience and empirical knowledge within the domain. The agile perspective focuses on how this knowledge will help us respond to change and even plan for it. This is helpful with the LSM since we expect the requirements to change when a pivot happens but still want to keep the overall functionality intact. This vision differs dramatically from classic software architectural practices popularized in the 1980s.

Lean Architecture	**Classic Software Architecture**
Gives the craftsman 'wiggle room' for change	Tries to limit large changes as 'dangerous' (fear change)
Lightweight documentation	Documentation-focused, to describe the implementation or compensate for its absence
People	Tools and notation
Collective planning and cooperation	Specialized planning and control
End user mental model	Technical coupling and cohesion

Figure 13 - Comparison of Lean Architecture and classic software architecture (Coplion 2010, p.5)

Traditional software architecture embraces engineering concerns very early and very strongly. Lean architecture on the other hand is about form and while the system should still obey the same rules that applies to engineering, they follow actual experience instead of scientific hypotheses. This means that developers should rely on experience and empirical knowledge to adjust the system when the requirement changes. However, it does not mean that Agile or Lean gives developers the power to completely ruin the system form as they like but rather honor the value of adaptation. Classic architecture on the other hand tends to be fearful of big changes, only incremental changes done to already existing artefacts are acceptable. Additionally, in the combined Lean and Agile approach, you can reduce risk by encouraging new forms in the parts that are likely to change. This allows the selected parts to be more adaptable to change since they are not filled with premature structure (Coplion 2010).

Creating a LSM architecture

Coplion (2010) has defined 18 different techniques for creating architectures in extremely uncertain- and constantly changing environments. We have selected the once we think are the most relevant concerning our research question.

Technique 1

Focus on the essence of the system form (what the system is) without being unduly influenced by the functionality that the system provides (what the system does).

Figure 14 - Lean architecture, technique 1 (Coplion 2010, p. 89)

Separating *what-the-system-is* from *what-the-system-does* is very important according to Coplion (2010). Architects should focus on the system form rather than the specific functionality. Functionality is always subject to change in these very uncertain environments as new empirical knowledge is generated. Therefore, the goal is to create a form, which is constant throughout the entire process and as independent from the functionality as possible. However, this can be a very difficult task for software architects since it means that simply utilizing traditional architectural framework like MVC is not sufficient. Coplion (2010) suggests to add an additional layer of likeliness to change to MVC:

Technique 2

Separate the components of your architecture according to their differing rates of change.

Figure 15 - Lean architecture, technique 2 (Coplion 2010, p. 90)

A suitable metaphor for understanding this technique is the case of building a house:
A stone foundation is not very likely to change in the near feature whereas internal walls are much more likely to change and the floor covering may change even quicker. These different rates of change needs to be managed in the architecture. This is the same for software; if we know that this functionality is very likely to change it should be reflected in the software architecture. We should not only divide the program in these specific modules (MVC) but within each of these modules, we should add an additional layer, which separates elements depending on their different rates of change (Coplion. 2010).

Based on these technique we are able to construct a customized architecture specifically designed for the environment entrepreneurs face in a startup.

3) Methodology

In this section, we will describe which methods are being used for this thesis. We will start by explaining how we have structured our research and elaborate on the specific approaches and methods we are going to use. Then, we will describe which types of data collection has been chosen and argue why this is the case. We will end this section by defining how our data analysis phase is structured and why our findings will be valid and reliable.

3.1 Research approach and design

This thesis tries to solve some of the technical challenges a startup will most likely encounter when applying the LSM process. We want to explore how we can prevent these technical challenges and thereby create a better synergy between the LSM and software development techniques. We have chosen to structure this process in the following way:
First, we have made an in-depth analysis of the LSM theory discovering several challenges when viewed from a software development perspective. Secondly, we have researched different theory in order to create a set of guidelines to prevent these challenges. After having created these guidelines from theory we are going to evaluate how they work in practice by applying them to our case company Flopfile. We will take these findings and discuss how they can contribute to field of science.

In order to do this we will have to establish in more detail how we are going to approach the test and analysis phase. We will dive into our philosophy and research strategy in order to build a well-defined foundation for our research.

3.1.1 Philosophy

When deciding which research philosophy to use it is important to understand what world the research is part and which role the researcher is going to take during the research.

This thesis builds on a pragmatic philosophy combined with elements found in interpretivism. The purpose of the thesis is not to reach one universal objective truth that will solve all of the technical problems in a startup, but rather to gain a deeper understanding on how these guidelines can assist the individuals in this hectic environment. The reason for using a pragmatic approach is due to how they define knowledge: *"a pragmatist stance aiming for constructive knowledge that is appreciated for being useful in action"* - Goldkuhl (2012).

We have created a set of guidelines derived from different theories in order to prevent a set of known problems. To verify that these guidelines work in reality, we have taken a role as an observer. Previous literature have stated that a set of problems can arise when using LSM together with software development and therefore we will observe what happens to the world when exposing it to our guidelines. Our belief is that observing the world with our guidelines can provide credible data to whether the guidelines have prevented the known problems or not.
As a second philosophy we have included Interpretivism in order to get a deeper understanding on how the users view these guidelines - both in cases where the guidelines prevent the given problems as well as when they do not (Holden & Lynch, 2004).

The reason for not using Interpretivism as the primary philosophy is because our research cannot be justified solely by subjective meanings nor does the researchers role affect the results. When applying the guidelines to the startup the researchers role is objective making sure they are followed and observing the outcome for different situations.

Unlike when taking a positivistic approach this research cannot rely on large samples and statistical analysis since the prerequisites for each startup will be different and in the need of the researcher to analyze which differences affect which situation.

3.1.2 Inductive vs. Deductive

When describing which approach we are going to use we will have to know what outcome to expect. For this thesis, the logical answer to this question will be to use a deductive approach since we are going to confirm a set of guidelines and this is partly true. However, we are actually going to use both an inductive approach and a deductive approach.

First, we will have to identify challenges using LSM with software development and create a set of guidelines to prevent these. This is where the inductive research approach is necessary.

"Inductive research involves the search for pattern from observation and the development of explanations – theories – for those patterns through series of hypotheses" (Bernard, 2011, p.7)

Bernard (2011) argues that the researcher "observes" and searches for patterns in order to develop general theory. In our case we have "observed" challenges when using LSM from a software development perspective through others researchers findings. We then look for patterns in their research in order to create tentative hypotheses and derive our guidelines. We will not be creating an entire theory but believe for the purpose of this thesis that a tentative hypothesis will be sufficient.

The second step will be a confirmation of our guidelines using a deductive research approach.

"A deductive approach is concerned with developing a hypothesis (or hypotheses) based on existing theory, and then designing a research strategy to test the hypothesis" (Wilson, 2010, p.7)

As Wilson (2010) describes we will have to develop hypotheses and a research strategy when using a deductive research approach. Our hypotheses will be our guidelines that will describe our predictions of the outcome (based on theory) when applied in practice. We will then test and observe the actual outcome in order to evaluate our guidelines.

3.1.3 Research strategy

Our research strategy is very inspired by the design science research methodology. Design science research provides a set of analytical tools and techniques in order to perform research in Information Systems (IS). It focuses on the design of artifacts in order to both solve a problem and contribute to the field of science. Although natural science research methods are appropriate when studying how objects or phenomena in society interacts with each other, it becomes insufficient when studying organizational problems that require creative and innovative solutions. These problems are best addressed using the design science methodology which is a type of paradigm shift from the natural science research. (Hevner, 2004).

In its core, design science is the study of the design of man-made objects (artifacts) with the purpose of meeting certain goals. In our project we created these guidelines deducted from the available literature and introduced them in our case company Flopfile. We wanted to investigate how our artifacts (our guidelines) impact the startup process both on the IS and the people in the company with regards to solving the given problems. These artifacts are often framed in terms of an inner environment, an outer environment and an interface between these two. The interface is designed to meet the specified goals (solve the technical problems within LSM). The inner environment consist of all the components of the artifact and their relationships whereas the outer environment is the external forces and effects on the artifact (Vaishnavi & Kuechler, 2012). How the artifact behaves is then constrained by both the organization Flopfile and the outer external environment.

This means that the design of the artifacts can be thought of as the creation of an interface between the outer and inner environment. More specifically, it is the process of designing and validating an artifact between our case company Flopfile and the external startup market, so that the given technical issues can be avoided. It is the knowledge derived from performing this mapping we can use to both solve the given problems and contribute to the field of science.

Our design science process is inspired by Vaishnavi and Kuechler (2012), it was important that knowledge contribution was a key focus on each of the stages in the process.

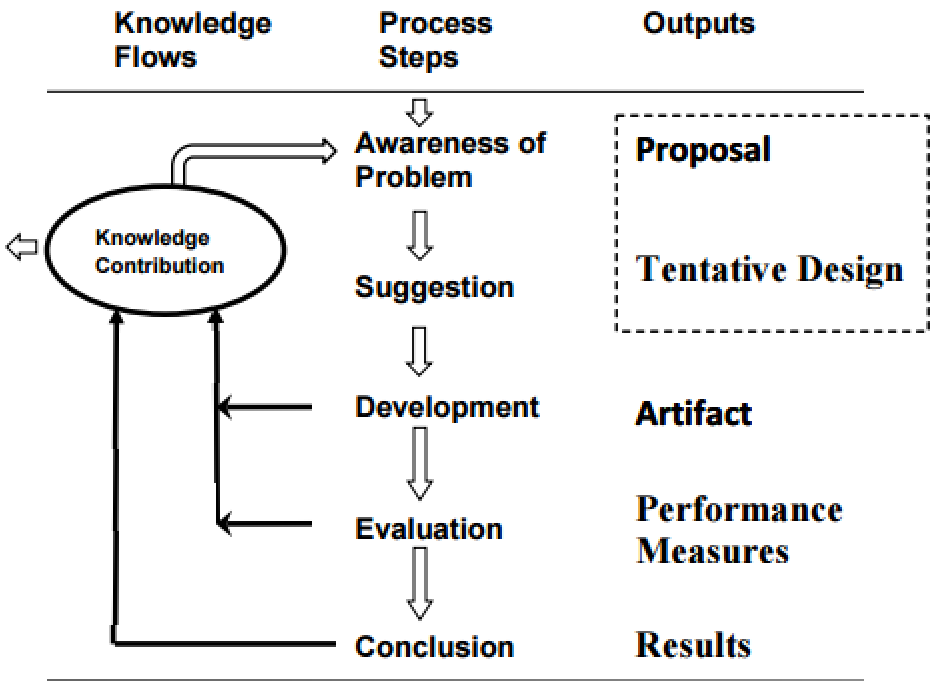


Figure 16 - Illustration of a design sceince process (Vaishnavi & Kuechler, 2012)

Awareness of Problem

In order for the research to be relevant, it must start with creating awareness of a problem. Our awareness of the technical issues within LSM came from multiple sources; external authors who have been criticizing the theory as well as the creators themselves (mostly Eric Ries) mentioning technical issues. However, the concrete details of these problems was not given beforehand. We first had to define these problems ourselves based on the literature. Using this inductive approach in the initial stage, we were able to define exactly what the problem is. The output of this stage was a list of possible technical challenges you encounter in LSM.

Suggestion

When the problem is defined, possible solutions can be suggested. The awareness and suggestion phase are very connected, which is why there is a dotted line between the two outputs in Figure 16 (Vaishnavi & Kuechler, 2012). The proposal would include a tentative design of the solution and the performance of a prototype based on this design. The suggestion phase can be seen in our 'Technical methods' section. We created the frame of this tentative design by researching literature on how these technical issues can be handled in environments similar to ours. These solutions are not necessarily directly connected to the LSM but shares the same characteristics and creates a foundation to which the guidelines can be developed.

Development

The artifact can now be developed. Based on the tentative design it is now possible to implement a solution for the given problems. The artifact does not need to be in a physical form but can easily be a piece of software or maybe an abstract concept.

The development phase is corresponding to the creation of our guidelines. Based on the surrounding theory on how such problems can be avoided, we were able to implement a list of specific guidelines that should prevent these problems.

Evaluation and conclusion

When the artifact is finished, it has to be evaluated according to the criteria stated in the proposal. Appropriate data collection methods must be chosen and the whole process should be thoroughly documented. Any slight deviation from the expectation should be carefully noted and explained. (Vaishnavi & Kuechler, 2012)

Since the study is of confirmatory nature there should exist hypotheses about the behavior of the artifact and the surrounding environment before starting the evaluation. However, it should be recognize that in design science it can be difficult to predict exactly what is going to happen:

"Rarely, in design science research, are initial hypothesis concerning behavior completely borne out."
(Vaishnavi & Kuechler, 2012, p.8)

Selecting the evaluation approach

Evaluation in design science research is a key activity since it provides feedback for further development. Even though the research within this area is still somewhat limited, John Venable *et al* (2014) have developed the FEDS (Framework for Evaluation in Design Science), which provides different strategies for doing evaluation in design science. We have used the framework in order to better explain what type of evaluation we did and why our results are valid with this method.

In order to choose an appropriate evaluation method you have to consider two important aspects; the functional propose (summative or formative) and the paradigm of evaluation (naturalistic or artificial).
The functional purpose of our thesis cannot be defined as being purely summative or formative since both aspects are represented to some extent. Naturally we need to assess our expectation about the guidelines which is of a summative nature, but it is also necessary to investigate how the users understand the artifact, why potential problem arises and how this can be solved which is formative. While the functional purpose is concerned about *why* to evaluate, the paradigm of evaluation focuses on *how* this should be done. John Venable *et al* (2014) uses a distinction between naturalistic and artificial evaluation in order to define the paradigm. Our study is purely naturalistic, as we want to explore how the design artifacts performs in its real environment. Due to the extreme uncertainty on the environment in startups it would not be relevant to do artificial evaluation.

With these properties, the FEDS suggest to use the *'Human Risk & Effectiveness evaluation strategy'* in order to effectively evaluate our design artifact. This strategy starts with formative evaluations in the early stages and quickly moves to more naturalistic formative evaluations. Towards the end, the strategy will use summative evaluation in order to assess to effectiveness of the artifact. This means that we will have to define several separate evaluation episodes, which ultimately ends which a summative assessment of the artifact:

1. Formative evaluation: Initial interviews with Flopfile about their company, the startup environment and technical problems related to the guidelines.
2. Naturalistic formative evaluation: The guidelines will be introduced in Flopfile, which they will use in their own natural setting. Here we will conduct various observation sessions in order to evaluate the artifact while it is in use.
3. Summative evaluation: The process will end with an evaluation in order to judge if the guidelines performed according to the initial expectations.

Contribution to knowledge

It is important to discuss what exactly the output of the thesis will be and how this will contribute to the field of science. Hevner & Gregor (2013) discusses different types of contributes in design science research depending on how abstract, complete and mature the knowledge is. The first level is a situated implementation of an artifact, this is when an artifact is transformed into a material existence that is fitted for a specific situation. The next level is more abstract and is defined as *'knowledge as operational principles'* (Hevner & Gregor, 2013). This can be overall methods, models or design principles that is designed to solve problems in the relevant domain. The third and highest contribution type is the design theory itself. A design theory is normally very abstract and contains knowledge about embedded phenomena, which is derived from extensive empirical data.

Since this experiment is only done on one company, it is not possible to generalize the results and create a full proven design theory. The purpose of this thesis is therefore to explore the effect of these guidelines and create a foundation for future research in this area in order to contribute to the field of science. Therefore, the output of this thesis and our contribution to knowledge will be a nascent design theory or a *'level 2 contribution'* according to Hevner & Gregor (2013).

3.2 Data collection

In relation to our choice of research method we will use a qualitative approach to collect our empirical data. According to Hair et al. (2011) it is the nature of the study and its objectives that determines which kinds of data is needed and thus which data collection methods are suitable for the thesis. Since we need to gather, information on how the technical aspect in LSM changes when these software-related guidelines are introduced, we deem the qualitative research approach to be most appropriate in this study (Newman & Benz, 1998). We need to understand the underlying reason and motivations of how these guidelines actually works with the LSM process and which affect it has in our case company Flopfile.

Although a quantitative approach could potentially provide some useful data it would not be appropriate in our context. Quantitative research focuses on quantifying data in order to generalize results from a sample group to the population. In our case, this would require multiple companies, which all had to implement our guidelines in their startup process and present the data to us usually in form of a questionnaire. Another drawback is the fact that you do not have the opportunity to monitor the process and talk to people face to face on how this experiment affected their business.

The collected data can be classified as primary data as it was collected only for the purpose of the study (Churchill, 1983). It consists of various observation sessions with comprehensive journal keeping and a number of qualitative interviews. We chose to do both interviews and observation sessions together as they each have their own strength and weaknesses and by combining these two types of data you gain a better understanding of how the reality actually is.

3.2.1 Observations

In order to document how Flopfile carried out the LSM process it was necessary to conduct a number of observation sessions. These sessions allowed us to gain a close look at how the process actually was carried out and what the reactions were in their own comfortable environment. This helped us to reflect on the experiences and ultimately understand the underlying reasoning (Wellington, 2001).

In order to keep our observations consistent, a research journal was kept during each session. This meant that we could capture key events right when they were happening instead of later where you run the risk of changed perception since time has passed by. Thus, the journal contained a description of what happened that particular day without any post analysis or reflection.

The second part of the journal was centered around informal and spontaneous questions we would ask about the things that were happening right now. We needed to understand peoples motivation and reasoning about the process and the artifact right when it was happening, and if we were to wait for the more formal and structured interviews we might receive a different answer.

It was important that we, the observant was mostly unbiased during the observations in order to not influence the results. However, we also needed to make sure that our guidelines were implemented correctly and the LSM process was followed which meant that it was impossible for us to remain completely neutral. When these kinds of situations occurred we needed to understand the reason before correcting it. Maybe one of the guidelines was worded in such a way that misunderstandings could happen or maybe the results simply differed from what we was expecting. This knowledge was crucial in order to perfect to guidelines and reflect on its effect.

3.2.2 Interviews

Semi-structured interviews were conducted with people within the organization about the technical challenges when using LSM and the associated guidelines. It was important to capture the insight and experience of the people actually doing the experiment - the employees of Flopfile.

The questions asked were similar to the once asked during the observation sessions, however these were more carefully worded and the people had time to really reflect on the question. The questions were based on the knowledge gained in the previous observation session, things that were unclear, motivations for specific actions and general opinion on the guidelines and its effect.

We did follow Blomberg & Giacomi's (1993) notion of open-ended semi-structured interviews which meant that our agenda in the interview was not set in stone but rather designed to allow the participants to shape the discussion. We recognized the fact that since the participants had first hand experiment with the implementation of our guidelines, the relevant topics might not be what we initially thought (Blomberg & Giacomi, 1993).

3.3 Data analysis

This section will describe how we have chosen to analyze our qualitative data gathered from Flopfile in order to reach our overall conclusion for this thesis. We will elaborate on the methods used and why these methods are appropriate for our thesis.

3.3.1 Organizing data

Once the observations and interviews were conducted we chose to organize all our qualitative data into categories. The data organizing process consists of four primary actions; data transcription, data translation, data cleaning and data labeling (Newman & Benz, 1998).

For this thesis data transcription and data translation were performed as parallel activities as suggested by Kvale and Brinkmann (2009). The non-verbal communication were chosen not to be included since we considered the verbal communication sufficient for the purpose of this thesis.

Data cleaning was performed to clarify the meaning of the sentences and make small "refinements" for both interviews and journal in order to improve the readability and understandability (Wellington, 2001).

After the data cleaning process we performed data labeling (also known as data coding). Newman & Benz (1998) argues when using a deductive approach it is allowed to use the predefined research questions to group the data in order to look for similarities and differences. In addition Crabtree & Miller (1999) states the importance to know your biases and preconceptions to avoid very subjective interpretations. As we have created categories in advance for our hypotheses (our guidelines) we knew we had to focus on being as objective as possible and not conclude anything too fast.
We have also focused on exhausting the data as much as possible by trying to account for all the text in the interviews and all the parts in the observations as recommended by Crabtree & Miller (1999).

After having structured our data we had created the foundation for evaluating our findings.

3.3.2 Evaluating data

When using an explanatory framework for analyzing your data the results can be used to confirm or reject the hypotheses. In this thesis we have chosen to evaluate how well our guidelines performed in Flopfile. Our data were entered in a scheme together with the guidelines to gain a comprehensive overview of which data parts that contributed to the confirmation or rejection of our hypotheses. According to Engel & Schutt (2009) the data can be viewed from three perspectives when analyzing it. These perspectives are as follows (Engel & Schutt, 2009 p. 295)

1. "When the researcher reads the text literally, the focus is on its literal content and form, so the text “leads” the dance"
2. "When the researcher reads the text reflexively, the researcher focuses on how his or her own orientation shapes interpretations and focus. Now, the researcher leads the dance."
3. "When the researcher reads the text interpretively, the researcher tries to construct his or her own interpretation of what the text means."

These three perspectives are important to consider when evaluating qualitative data in order to try avoiding a wrong conclusion (Engel & Schutt, 2009). We have tried taking this into account by iterating through the evaluation using the different perspectives separately.

3.4 Validity and reliability

In order to ensure the conclusions drawn from our research are valid and reliable, we will have to specify what exactly ensures validity and reliability. According to Yin (2008) the degree of validity and reliability in a case study can be described by testing four components; Construct validity, Internal validity, External validity and Reliability.

3.4.1 Construct validity

Construct validity tries to ensure that what is going to be measured actually is being measured. The focus in construct validity is on exposing and reducing subjectivity. Yin (2008) argues that construct validity can be ensured using three different procedures; using multiple sources of evidence, establishing a chain of evidence and having the key informants reviewing their interviews.

During our case study research multiple sources of evidence was used since we observed and interviewed each of the three co-founders of Flopfile individually. The chain of evidence was kept using our journal where each step was described thereby creating a structure around our research. Lastly we ensured that all interviews were reviewed before being analyzed.

3.4.2 Internal validity

Yin (2008) describes internal validity as establishing a causal relationship between findings where certain conditions are shown to lead to other conditions. In our case we are trying to show that using our guidelines will lead to certain results. This can be ensured using two different procedures; doing pattern matching and explanation building.

Our researched is highly structured around matching different patterns when searching for evidence. Especially because we want to determine that our results are caused by our guidelines. In order to accomplish the best results regarding internal validity we focused on explaining what could lead to our finding in order to justify them.

3.4.3 External validity

External validity tries to validate whether or not the finding of the research can be generalized for the specific domain (Yin, 2008). In order to ensure this Yin (2008) describes two procedures; using replication logic in multiple case studies and using case study protocol.

In our study, external validity will be perceived to be rather low since we have only conducted a single case study. It is possible that some of our finding may be generalizable for other similar companies but this can not be stated with high probability. Although we have documented our specific procedure for our research and thereby creating the possibility to extend our research and improve the external validity.

3.4.4 Reliability

When a study is able to demonstrate that the operations (such as data collection) can be repeated with the same results, reliability is achieved (Yin, 2008). This can be achieved through appropriate record keeping and documentation of procedures.

In order to ensure reliability we have created a journal and interviews. This helps us documenting our process for further use. Afterwards we have analyzed our finding and described how we have reached our conclusions. Therefore we perceive the reliability of our finding to be high even though we are working in a rapid changing environment.

4) Creating the guidelines

We have now analyzed the core principles of LSM, defined the technical challenges when applying this methodology in software projects and explored technical theory used in similar environments. Therefore, we are now able to construct a set of guidelines in order to minimize the risk of these technical challenges occurring. In this section, we will describe the creation process and discuss the different guidelines in depth; what purpose do they serve, how have they been derived and what is the expected effect when applied in practice.

4.1 The creation process

Once we had identified a set of challenges the next step was to discover how these challenges could be prevented in similar environments. By doing this, we would be able to draw parallels between different authors way of handling these challenges in other environments and distinguish how it would differentiate from environments using LSM.
Primarily we analyzed software environments with a high degree of uncertainty which is similar to the LSM. In order to do this we used the LSM principles and compared these to well-known software development methodologies.

After having researched how to prevent these challenges, it was time to create specific guidelines customized for the LSM process. We have created a scheme that identifies which theory has been used to create each of our guidelines. This scheme is shown below.

Technical challenges	Theory	Guidelines
Eliminate waste: Very difficult to eliminate software waste in environments with high uncertainty **Possible downside:** • **Unnecessary time and money spent**	Software complexity concept (Schwaber, 2010; Appello 2008) Prescriptive and adaptive software methodologies (Kniberg & Skarin, 2010; Norrmalm, 2011; Boehm, 2004; Appelo, 2008) Scrum (Schwaber, 2001) Kanban (Schwaber, 2010)	**Guideline 1**: Minimizing waste in your software

	Lean software development principles: "Decide as late as possible" (Cobb, 2011; Poppendieck, 2003) Minimal Viable Product (Blank, 2006; Ries, 2011; Furr & Ahlstrom, 2011)	
Architecture: LSM Devalues software architecture when focusing on MVP **Possible downsides:** • **Spaghetti architecture - when changing one component influence other.** • **Scaling becomes impossible**	Software complexity concept (Schwaber, 2010; Appello 2008) Prescriptive and adaptive software methodologies (Kniberg & Skarin, 2010; Norrmalm, 2011; Boehm, 2004; Appelo, 2008) Scrum (Schwaber, 2001) Kanban (Schwaber, 2010) Lean Software Development (Bjørnvig, 2010; Coplion, 2010; Ballard, 2000) Lean architecture (Coplion, 2010) MVC (Reenskaug, 2003; Deacon, 1995; Burbeck, 1992)	**Guideline 2**: Create a lean architecture
Innovation: LSM hinders true innovation when focusing on MVP **Possible downside:** • **The startup loses a competitive edge**	Scrum (Schwaber, 2001) Kanban (Schwaber, 2010) Lean Software Development principles: "Amplify learning" & "See the whole" (Cobb, 2011) Validated learning (Ries, 2011)	**Guideline 3**: Innovation as a development activity

The scheme shown above describes which literature that has been used in order to create these guidelines. It is important to recognize that not all the literature contributes directly to the actual guideline, but instead to the overall creation process in order to provide enough knowledge to be able to customize the guideline for LSM and the relevant software development methodology (SDM).

If we take a look at "Software complexity concept" by Schwaber (2010) & Appello (2008) this section has been used to get an understanding on which kinds of SDM's might be relevant for this thesis since there exist numerous SDM's used for different kind of purposes. After having narrowed down the number of SDM's in our search field we analyzed which SDM would be the best fit for the LSM whether it was a more prescriptive- or adaptive methodology. We discovered that Scrum (Schwaber, 2001) and Kanban (Schwaber, 2010) would be the optimal choice of SDM when using LSM. This gave us enough knowledge to customize our guidelines specifically for these SDM's.

When having established our base knowledge we used the additional literature in the scheme in order to tailor a solution to each of the technical challenges. Further description on how each of the three guidelines has been created is given in the following subsections regarding the guidelines.

4.2 Guideline 1: Minimizing waste in your software

Working in a startup environment is often very unpredictable and radical changes are made more often than in well-established companies. This can make the software development process more complicated and time consuming since you run the risk of redoing work. In order to make the environment more predictable we can postpone decision-making.

We saw in the technical challenges that big changes in software development often result in modifying the architecture and even sometimes scrapping working code. Since LSM focuses on eliminating waste, we need some specific guidelines in order to do this from a software development perspective. In order to help solve this we used one of the Lean Software Development principles: *"decide as late as possible"* (Cobb, 2011; Poppendieck 2003). According to the theory when you avoid locking in decision until you have enough data to support it you build a capacity for change which automatically reduces the amount of code needed to be reworked when these changes occur. That means that you should never start doing any programming until you can justify spending resources on a functional MVP.

4.2.1 The guideline

This guideline is two-folded describing the decision making process before you start developing your solution as well as during the development process.

- Keep your options open and avoid locking in decisions. Do not start implementing specific features and functionality prematurely. You should always delay decisions about the system as long as possible until you have sufficient empirical knowledge to know which features are actually necessary.
    - A functioning software prototype should never be implemented until you have sufficient empirical data to know this type of product is something customers actually want. In order to do this you should start with a more simple and inexpensive MVP that does not require any programming. This would typically be some basic wireframes you present to possible users

The overall idea with this guideline is to delay decisions as much as possible during the development phase. By delaying decisions until sufficient empirical knowledge has been collected you minimize the risk of code scrapping due to wrong assumptions.
The sub part of this guideline operates with an initial prototype. By creating a MVP/prototype before you start programming helps you gain initial feedback from the customers. This will minimize the risk of creating software waste during the startup phase.

Expectations

When applying this guideline in practice our expectations are as follows:

- Changes in the software will not result in large code scrapping.

4.3 Guideline 2: Create a LSM architecture

It is of paramount importance that you do not neglect the use of proper software architecture when using LSM. You have to build a capacity for change in the software. In order to do this you must first focus on the system form rather than the specific functionality. The system form is how you develop the overall system using empirical knowledge. This can be achieved with a customized software architecture specifically designed for LSM.

This guideline is derived from the mismatch between LSM and traditional software architecture. Several authors has criticized LSM for devaluing architecture and described how traditional architecture was not optimized for LSM. Therefore, it would not be sufficient to simply create a guideline saying that you should still do software architecture. Instead, we wanted to define a custom architecture specifically designed for LSM in order to create an incentive for using architecture when you know changes in the system are bound to happen.
We recognized that we did not have the experience or data to justify a completely new software architecture, which is why we decided to take the world's most used architectural pattern MVC and optimize it for new startups. This way we are still utilizing the benefits that comes with MVC and the fact that many programmers are already familiar with this pattern. In order to optimize this for new startups we used Coplion's (2010) techniques for creating architectures in extremely uncertain environments. Since Coplion's techniques are not very concrete like *"focus on the essence of the system…"* and *"separate the components of your architecture according to their rate of change"* we needed to analyze how this could be done in practice and how this could be implemented in alignment with the MVC pattern. We decided to introduce another layer of separation to the MVC. This layer would be the separation of the component's rate of change as described by Coplion (2010). By reducing the correlations depending on how likely components are to change, it reduces the impact on the rest of the program when these changes occur. This allows the programmer to have more "wiggle room" in the system, which is a key principles in a lean architecture (Coplion, 2010).
We also wanted the architecture to defer actual implementation so you can focus on the description of relationships in the start. This is achieved with the top layers (the ROC-1 components) see Figure 17. By defining these components first, you focus on how the three separate modules (MVC) should communicate in the specific system without worrying about specific features yet.

With this model we have incorporated most of the essential principles when creating an architecture in extremely uncertain environment we saw from the theory, while still keeping the core structure of MVC intact.

4.3.1 The guideline

- You must build your system around the *'LSM software architecture'* which is designed in order to allow for changes in requirements without redoing the entire application. You start by separating your components in Model, View and Controller and then separate each of these modules according their rate of change. It is important to keep the dependencies as shown in the Figure 17, each components should only know itself and the module above it.

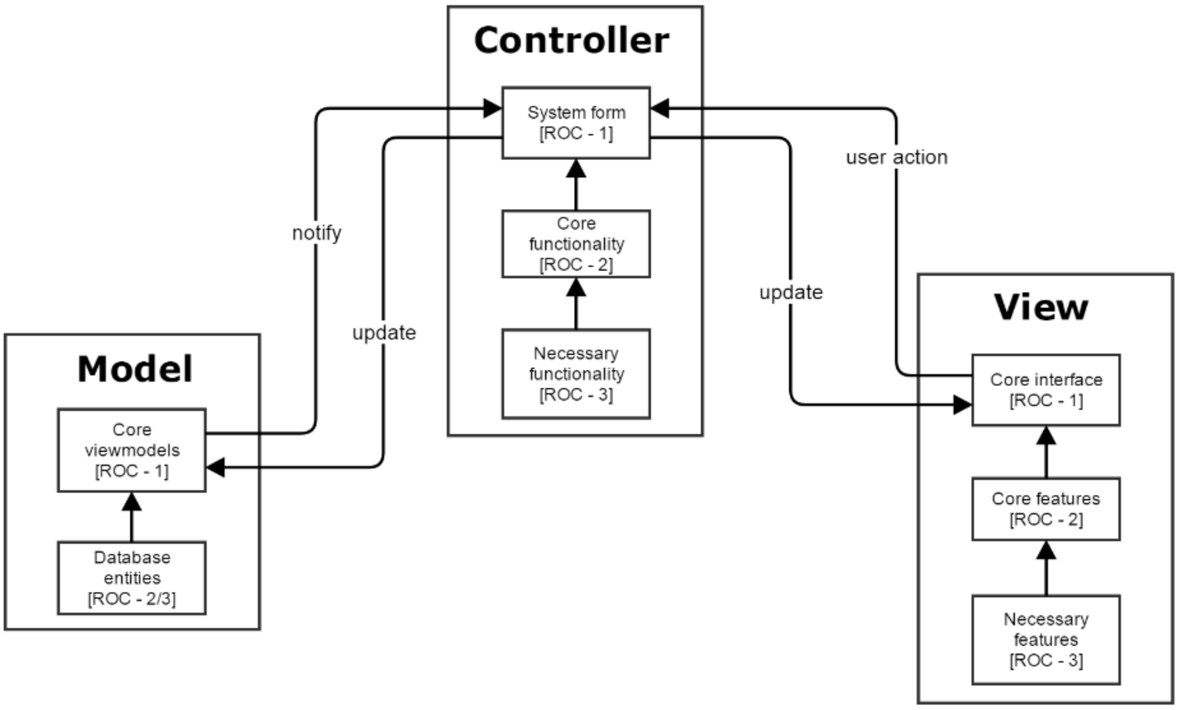

Controller
System form [ROC - 1]
Core functionality [ROC - 2]
Necessary functionality [ROC - 3]
notify
update
user action
update
Model
Core viewmodels [ROC - 1]
Database entities [ROC - 2/3]
View
Core interface [ROC - 1]
Core features [ROC - 2]
Necessary features [ROC - 3]

Figure 17 - LSM software architecture' – An architectural pattern designed for entrepreneurs using LSM (own creation)

How to use the guideline

The *'LSM software architecture'* builds on the classical Model-View-Controller architectural pattern and extends it in order to allow for further separation based on the rate of change (ROC) of the specific component. The ROC is an estimation of how likely this component is to change; a component with a high ROC should be implemented as an add-on, which can easily be changed or even deleted. The table describes how the ROC should be estimated.

Rate of change (ROC)	Description
1	Very unlikely to change, this is the overall *'system form'* that shapes the entire system. More concretely, this is a generic platform for communication with the other modules in order to avoid constantly changing each module every time a change happen.

2	Core functionality with strong empirical evidence that there is an actual need for by the user.
3	Features/functionality needed in order to build the current version of the system but are likely to change in a pivot.

In order to use this architecture effectively you need to be familiar with the basic concepts of the MVC architectural pattern (explained in section 2.4.2). You start by separating the data, the business logic and the visual interface like in MVC. Then you must separate each components within the three respective modules according to their ROC. It is important to recognize that the ROC estimation is always based on empirical data gathered and the nature of the component itself. Even though you think a specific feature is essential for the system - without sufficient evidence it should receive a high ROC. The dependencies are then split upwards so that it is only ROC-1 that can communicate with the other modules. This ensures that even though features change a lot, the communication and the separations of concerns is constant throughout the entire project. Changes in the program are then always handled within their own module. We have described each of the modules presented in Figure 17 below.

- Controller
    - System form (ROC-1) – This is the overall shape of the system (system form) that is essential in order to solve the problem. The responsibility of this module is to communicate with both Model and View. The user's request will always be send directly to this module. Then it needs to request relevant data from the model, process this data if necessary and send the information to the view. Because of the many dependencies with the other modules in the program, it is important that the probability of this changing is at a minimum. If components in this module where to change it would affect all the other modules.
    - Core functionality (ROC-2) – The core features of the program that is not very likely to change according to your empirical data. However, because of the extreme uncertainty you face in a startup if these features were to change it would only affect this module and the necessary functionality.
    - Necessary functionality (ROC-3) – Features necessary in order to build the current version of the system but are very likely to change in a pivot. Changes will only affect this module itself.

- Model
    - Core view models (ROC-1) – The core view models are not raw data itself, but rather a platform for communicating with the controller. Since the database entities are very likely to change, you want to establish a generic platform for commutation you can easily manipulate when changes happens.
    - Database entities (ROC-2/3) – This is the raw data, whether it comes with own local database or via an external API[3]. It is not necessary to separate your specific entities any further when you have your core view models.
- View
    - Core interface (ROC-1) – Like the models, the visual interface is extremely likely to change during the project. The core interface is therefore a generic shell for your program without any specific features in order to communicate with the controller.
    - Core visual features (ROC-2) – Like the core features from controller, this module include the visual components, which are essential to your program.
    - Necessary visual features (ROC-3) – The visual features needed to support the necessary features in the controller. Very likely to change.

As you can see in Figure 17, the higher the ROC a components has the less dependencies it has. The system form in controller does not depend on the specific features at all, it simply provides an interface of communication that the lower modules can access. In order for this kind of separation to be possible, you need to build an observable pattern where components can subscribe to different events in order to execute them. E.g. when a user access a webpage it will call an event to the system form in controller, the system form will then publish the event to all its listeners which will execute the specific command and send it via the provided interface to the View. This means that you can freely change, add or delete components in the *'necessary functionality'* module without affecting anything else in the program. Obviously if you need to add a new feature, you will also need to implement the corresponding visual components in View.

[3] API (Application Programming Interface) - a set of programming instruction in order to access external data via the web

This kind of dedication to a system architecture will naturally require more time in the initial phase than most programmers are used to, but when it is in place it allows for a new form of dynamic programming needed for LSM.

Expectations

We have the following expectation when using our architecture in practice:

- Before any actual programming, the entrepreneur will spent time defining how each module should communicate. This platform for communication will not change very much during the startup process.
- Changing, adding or deleting specific features will be easy and only require minimal rework in other areas of the program.

4.4 Guideline 3: Innovation as a development activity

When a startup focuses on rapid development as a core activity the priority of innovation is often downsized. The problem with downsizing innovation for a startup is that the company potentially miss out on opportunities for an extra competitive edge in their respective market. Therefore, entrepreneurs will have to build activities to focus on innovation without compromising the rapidity of LSM. This can be achieved with a specific focus on innovation as a development activity at the right time during a LSM cycle (Cohn, 2014).

This guideline has been created due to the issues Cohn (2014) describes regarding the well known development methodology Scrum. Seen from a software development perspective Scrum and LSM has many similarities like *'Minimum Viable Product'* and *'Iterate rapidly'*. Due to limited research on LSM in this area, critique from Scrum has been used in order to justify the need for a guideline regarding innovation.
According to the literature used in this thesis, programmers should consider different elements when developing software using LSM. This guideline has been derived mainly from the two lean software development principles *'Amplify learning'* and *'See the whole'* (described by Cobb, 2011) as well as the LSM principle *'Validated learning'*. *'Amplify learning'* and *'Validated learning'* both centers around creating and testing ideas in order to base decisions according to empirical findings. In addition *'Amplify learning'* suggest to communicate early and often with the other team

members. When combined with the *'See the whole'* principle, where the team should focus on viewing the solution as a whole, the foundation for the guideline is created.

4.4.1 The guideline

- During the LSM Build-Measure-Learn feedback loop, create specific activities that focuses on innovation. These activities should then be performed by the whole team as a group during the "Learn" and "Ideas" phase of the feedback loop.

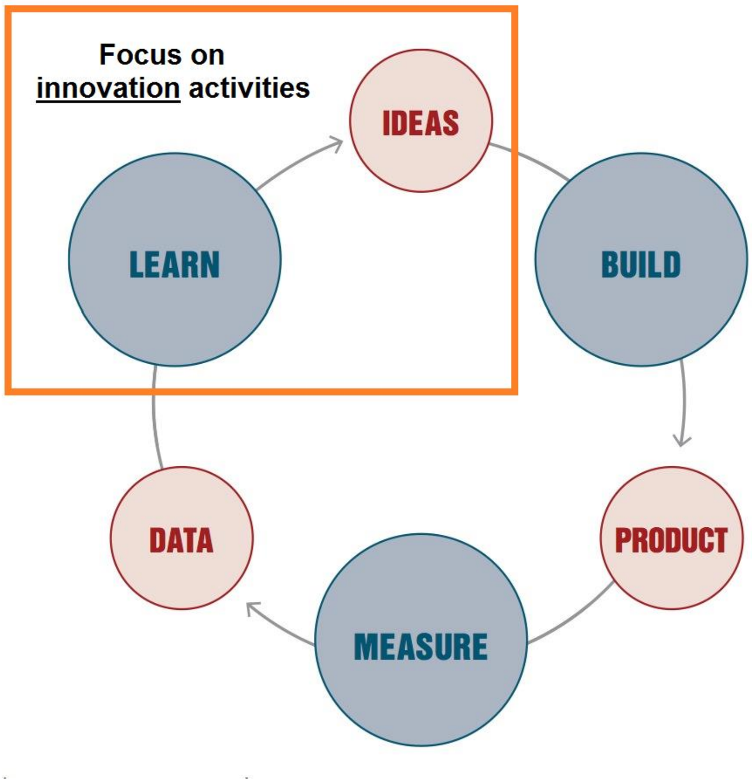


Figure 18 - Innovation inspired Build-Measure-Learn feedback loop (inspiration: Ries, 2011)

By focusing on innovation during the *'ideas'* phase the team can benefit from each others ideas before developing specific features - uniting the mindset of the team members and create a common understanding about the product. With a focus on innovation during the *'learn'* phase the team will have the opportunity to follow up on the last cycle (from an innovation perspective) and discuss whether or not certain product features have had the desired effect on the customers as well as how the next cycle can benefit from these findings.

Expectations

When applying this guideline in practice our expectations are as follows:

- Several innovative ideas for possible solutions/features etc. will be created due to each team members involvement in the idea generation process. This will give the team the ability to select their solution from multiple ideas and not just choose the first and cheapest solution.
- Each team member will end up having a good understanding of the product as a whole during the development phase including which parts are being created by the others.

5) The case of Flopfile

In this section, we have described our findings gathered from our case company Flopfile. We have divided it into three sub sections including; introducing Flopfile to LSM, the LSM process and evaluating LSM and the guidelines.

In the first two sub sections we have described our two months with Flopfile and elaborated on the process they have used including important decisions made during this process.

In the third sub section, we have collected all the data regarding our guidelines in order to easily grasp the most essential data concerning our guidelines. First Flopfile have evaluated how they have experienced our guidelines followed by a one-pager summarizing our findings.

5.1 Introducing Flopfile to LSM

In order to validate our guidelines we needed to gather empirical results from an actual startup company. To do this we found a case company called Flopfile who wanted to improve their chances of success and agreed to gain help from us since none of them had any real theoretical knowledge on how to build a company from scratch.

After having created our guidelines, it was time to visit Flopfile and make sure they got started properly. For the meeting we had planned to establish a baseline on how far Flopfile actually was in the process of creating their end product, provide them with an in-depth introduction to LSM and describe what our guidelines could do for them.

First, we did an interview with Kristian - one of Flopfile’s co-founders who gave us a description of the fundamental business idea behind the startup. He described how it all started from an everyday pain, he and the other co-founders faced when having multiple cloud storages and sharing files between them.
We wanted to discover how far they were in the process of creating an actual solution for this and if they had performed any prior testing to validate their vision. Kristian said they were still in the analysis phase and had not created an actual product yet. Prior to our meeting the had analyzed whether or not their product idea was even possible to create since it relies on several suppliers and their data availability. Kristian told us they had also performed some market research in order to determine existing competitors and whether or not there might be an interest in their product idea

from a customer perspective. Specifically they had performed two in depth interviews with potential customers, which both revealed a market potential for Flopfile's product idea.

After having conducted the interview with Kristian, we held a workshop for the founders of Flopfile in order to introduce them to LSM. During our workshop, we discovered that Flopfile quickly understood terms like MVP, validated learning and working iteratively. They told us they were quite familiar with the development methodology Scrum and that there were many similarities. Although they had never thought of using this terminology from a business perspective since they had only worked with it solely from a development perspective.

When we felt Flopfile understood the basic principles of LSM it was time to introduce them to our guidelines. We described how these guidelines were designed in order to help them create the best product in the most efficient way - similar to the core principles of LSM.

The workshop ended with us planning an observation session once every week for two months in order to follow up on how LSM and our guidelines worked for Flopfile. This way we would be able to analyze the good and bad parts of their process in order improve it in the long run.

5.2 The LSM process

This section will elaborate on how Flopfile has managed their process during our research period.

5.2.1 Vision

After the initial introduction with Flopfile we were able to define a baseline for how far in the development phase they currently were. Flopfile had not created an actual prototype/MVP yet but had performed a small market study to analyze their chances in the market. Therefore, Flopfile were able to adapt to LSM from the beginning without a heavy switching cost.

In this section we will elaborate on how Flopfile has specified the company's grant vision and broken it down into smaller initial hypotheses in accordance to the LSM process. Moreover, we will describe the process of creating a MVP in order for Flopfile to test their hypotheses and be able to build their future product based on empirical data.

Creating initial hypotheses

In order for Flopfile to create their initial hypotheses it was nessesary to specify their grant vision and define their specific customer segment. Each member of the team had an idea of what the initial hypotheses should be but the team had not discussed these ideas with each other. Due to guideline 3, the team chose to hold a session where they would brainstorm in order to use every team member's ideas and creativity to define the grant vision and break it down into smaller hypotheses.

"After you visited us the last time we sat down and read more about LSM and how you apply it properly in a company like ours. We also read your guidelines and decided we would start out by having a brainstorm session in order to get our minds united and try to become as innovative as possible." (Mads, Appendix B1)

The outcome of this brainstorm session was a definition of the grant vision broken down into four value hypotheses and two growth hypotheses.

- **Grant vision:**
  Creating a complete overview of files located at multiple cloud storages
  - Value hypotheses
    - **Initial hypothesis 1:**
      The program should include the following cloud storages as a minimum: Dropbox, Google Drive, OneDrive, Box and Amazon Elastic Cloud
    - **Initial hypothesis 2:**
      The program should include a single interface of all the user's files across his/her cloud storages.
    - **Initial hypothesis 3:**
      The program should facilitate login via facebook for easy login
    - **Initial hypothesis 4:**
      The program should contain a drag and drop file sharing feature between multiple personal cloud storages.
  - Growth hypotheses
    - **Assumption 1:**
      Early adopters can be found in universities and schools as well as online

blogs (tech enthusiasts). Moreover, different companies using cloud storages can also be targeted later.

- **Assumption 2:**
  Flopfile can increase their customer base through word of mouth because existing customers and the new customers will get a referral bonus. Later on online ads will also be a possibility for growth

Creating a MVP and test plan

After the initial hypotheses, Flopfile decided to create a MVP in order to validate them. According to LSM, the entrepreneur should always test the initial hypotheses prior to any form of product development in order to ensure the product features selected for implementation are based on empirical evidence. Mads (one of Flopfile's co-founder) told us in order to follow the first guideline regarding a non-funtional MVP they decided to create a design mockup. A design mockup was chosen for two reasons: One of them was that Flopfile created a design mockup in order to visually represent their ideas since the product might be a little difficult to understand for some customers. The second reason was that at the same time they could discover which features should be implemented for their second MVP where they would start the coding process.

Flopfile decided to split up this next process. Two of the team members would be working individually on mockup designs while the last team member would focus on creating a plan for validating the initial MVP and their hypotheses.

Flopfile chose to structure their test plan in the following way:

- Locate 20 potential early adopters among university students (test persons)
- Contact them face to face asking if they want to participate in the test
- Measure the hit rate (How many said yes and no)
- Perform the test
  - Ask the test persons about their cloud storage habits
  - Describe the potential product
  - Provide a demonstration on how the product might look like (mock up)
  - Observe how the customers react to the different designs
  - Make a follow up interview about the customers perception
  - End the test with two questions:

    - "Would you be interested in buying this product once it has been created?"
    - "Would you like to participate in future product tests?"
- Evaluating the results

Validating initial hypotheses

After having completed the mockups and the test plan for their first MVP it was time for Flopfile to carry out the test. First, they would have to find 20 potential early adopters among university students. Flopfile decided that the IT-University of Copenhagen would be the best place to find early adopters interested in their specific product. Therefore, they went to the ITU and chose people they thought might be interested until they had 20 test subjects that agreed to answer a few questions. Flopfile discovered that the hit rate was 74%, which was above the 50% that LSM describes as necessary. After having found the test subjects, they executed their test plan and got the following results:

Initial hypotheses	**Evidence showed**
A MVP must include the following cloud storages in order to be functional: • Dropbox • Google Drive • OneDrive • Box • Amazon Elastic Cloud	The test was conducted on 20 people within the target group 90% used cloud storages 50% used multiple cloud storages 35% used both Dropbox and Google Drive 25% used both Dropbox and Onedrive None of the asked participants used Box and/or Amazon Elastic cloud
A MVP must include a single interface of all the user's files across his/her cloud storages	65% of the test persons used cloud storages for different purposes (work, school etc.) so while they thought one single interface is a good idea, it was not a must have.
A MVP must include a login via Facebook	90% of the test persons had a Facebook account

	85% of the test persons having a Facebook thought it was a very smart idea. Although they would still use a smart application without the Facebook login.
A MVP must include a drag and drop file sharing feature between multiple personal cloud storages and multiple friends.	80% of the test persons thought it sounded smart. Although 50% of them was not sure if they were going to use it or not.

When Flopfile asked whether the test subjects would be interested in buying the product once it has been created, 30% answered "Yes" immediately, 45% answered "Maybe, depending on the price" and 25% answered "No". In addition 60% agreed to participate in future product tests.

After the first test it was time for Flopfile to evaluate the resulst, what they had learned, what to do in the future and whether to pivot or persevere. They all quickly agreed it went well and that they would persevere based on their findings.

"We all think it went really well for a first test and we are glad that we have done it. Even though we have chosen to persevere we have learned a lot from this test that we will take into consideration during our next LSM cycle. One thing is that we have tested our initial hypotheses, another thing is that we now have a lot of potential early adopters already in the pipeline, that we will be able to get feedback from in the future." (Mads, Appendix B3).

5.2.2 Steer - first iteration

The second phase of the LSM process centers on further improving the foundation for growth. The first MVP test illuminated that the issue of organizing files across multiple cloud storages was an actual problem. Since this initial hypothesis was validated, it means an actual functional solution to these problems could be implemented. This phase was divided into 2 sub-phases which would loop thus improving the product incrementally. The first iteration will be centered around developing the first physical MVP and especially defining the architecture.

Using the LSM architecture

Based on the discussed literature, the use of proper software architecture is a key success factor in any startup with a software product. In accordance to guideline 2, it was necessary to fully define the architecture before any programming could be made. The programmers of Flopfile were already quite experienced with the MVC pattern and therefore it was only the rate-of-change separations that was foreign to them. Using the data gathered from the initial tests, the team had an overall understand of which features would be necessary for the first functional MVP. Flopfile used this data in order to shape the overall system. However, that the amount of data was still very limited, the features would be likely to change in future iterations. The following functionality needed to be implemented for the second MVP:

- Login/register user
- Attach a single Dropbox, Google Drive and Onedrive account to your profile
- View your files and folders
- Basic directory operations like add, delete and rename.

Flopfile chose that sharing files would not be a necessary functionality for the first MVP due to the initial tests. Although the tests did show that the users found the feature useful, there simply was not enough evidence to justify implementing this feature.

"We knew that we had to delay decisions as long as possible and therefore we chose not to implement the sharing functionality since people were more focused on the file overview in our interviews. However, we knew that it could potentially be needed in the future" (Kristian, Appendix B3)

Defining and creating the architecture was the next step. In order to avoid locking in decision early and thereby minimize the risk of redoing a lot of code due to changes, they followed guideline 2, which was the LSM architecture. The first step was to identify the three modules - model, view and controller.

Model (the data)

The data Flopfile uses is very untraditional compared to many other online applications. Normally you have one big database where you can store and fetch all the information needed. However, in Flopfile the data was two-split; they have their own local data but also several external data sources.

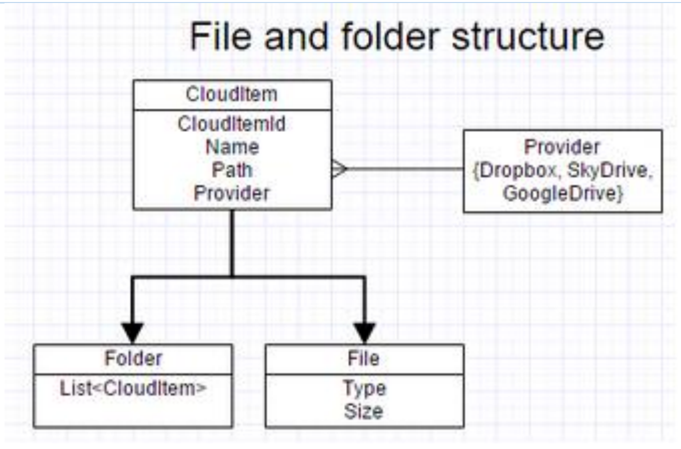
- Local data – This would be information about the registered users. Things like name, account name, emails and password are all necessary in order to recognize registered users. This data would be stored in their own local database. The ROC is very high since attributes about a user could easily be added or deleted.

- External data – In order to display a user's personal cloud storages files it was necessary to fetch this information from the relevant API. This meant that in order for the website to display the files and folders in the user's Dropbox, it was necessary to access Dropbox's public API with the correct credentials.

The external data was a very interesting case. In order to use the LSM architecture correctly you would have to split the view models and the actual database entities in separate layers, as described in the guideline. The Flopfile team knew that the exact number of supported cloud providers were very likely to change due to guideline 1, so they had to create an architecture that allowed additional providers to be added in the future without breaking the program. They did this by creating one single external view model structure for every cloud provider, so the data structure would be 100% the same regardless of the specific provider:

"We knew that our cloud storages had a high ROC since there could easily be added or even removed some as the time went on. Therefore, by using the file and folder structure to communicate with the controller we can add these without affecting the rest of the program - at least that's the idea."
(Kristian, Appendix B3)

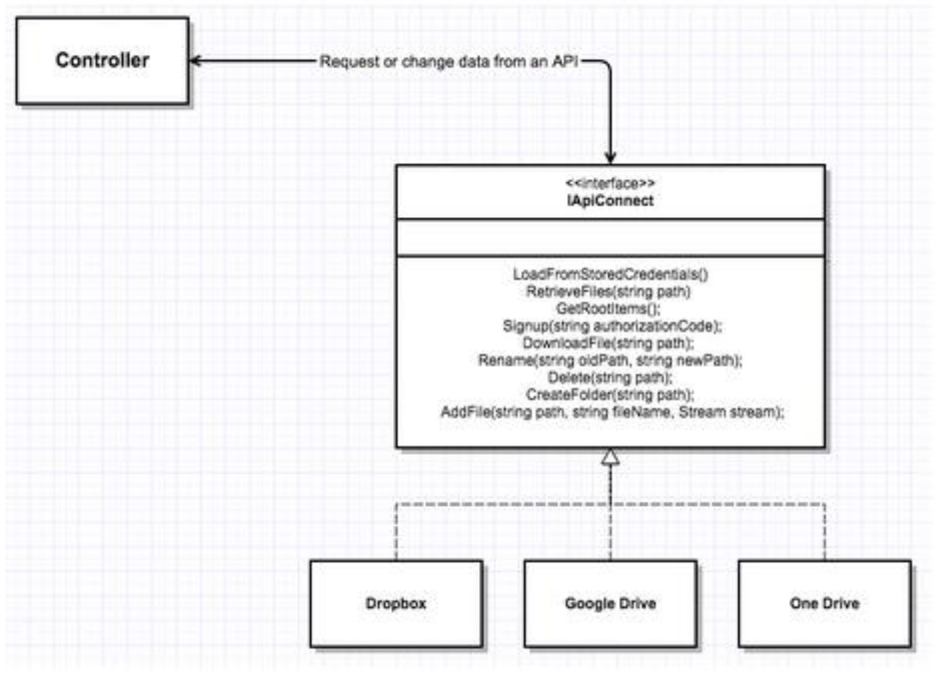
Module	How it was implemented in Flopfile
Core view models (ROC-1)	The core view models determined how the controller and the model communicated. The data structure for a user's files is illustrated on Figure 19. Each item in a user's cloud storage is categorized as a 'clouditem'. Using an inheritance structure, a cloud item can be either a file or a folder. The type 'file' contains information about the specific files, like type and file size whereas the folder contains 0 to many clouditems. This is essential a recursive data structure as a folder can contain folders, which can also contain folders and so on, just like in a normal file directory. The attributes in clouditems (Name, path etc.) is information that is shared between both files and folders.

	 File and folder structure CloudItem CloudItemId Name Path Provider Provider {Dropbox, SkyDrive, GoogleDrive} Folder List<CloudItem> File Type Size *Figure 19 - Data model structure for Flopfile (Appendix B3)*
Database entities (ROC-2/3)	This will contain the API access to the relevant cloud providers, which will fetch and store the data as needed.

Controller

The controller module was separated into three sub-modules depending on the components ROC in accordance to the LSM architecture.

Module	**How it was implemented in Flopfile**
System form (ROC-1)	The system form in Flopfile work as an endpoint which will handle certain requests from the View. When a user clicks a button it will send an asynchronies request to the system form. This sub-module has no information about the specific request but just publishes the request and sends it back to the view once it is handled.
Core functionality (ROC-2)	The core functionalities are illustrated in Figure 20. These functions are key features for the program to work. Here we have 'RetrieveFiles', which will return a user's files, download file, rename, delete etc. The interface *'IApiConnect'* works like a contract for all the cloud providers. All of the cloud providers must implement this interface and implements all of its operations. This allows new cloud providers to be added without affecting any other components in the program.

	 Controller
	Request or change data from an API
	<<interface>>
	IApiConnect
	LoadFromStoredCredentials()
	RetrieveFiles(string path)
	GetRootItems();
	Signup(string authorizationCode);
	DownloadFile(string path);
	Rename(string oldPath, string newPath);
	Delete(string path);
	CreateFolder(string path);
	AddFile(string path, string fileName, Stream stream);
	Dropbox
	Google Drive
	One Drive *Figure 20 - Controller structure for cloud providers in Flopfile (Appendix B4)*

Necessary functionality(ROC -3)	The 'necessary functionality' contains the specific implementation for all these methods. E.g. If the Dropbox module receives a 'Downloadfile' request, the actual executing will happen here. All of these components needs to be kept separated because of the ROC, since the implementing of the method depends on the specific cloud provider.

View

The components of the visual interface of Flopfile was also organized in 3 different sub-modules.

Module	How it was implemented in Flopfile
Core interface (ROC-1)	The communication module in the view was also seperated from any program specific implementation to keep the separation to a maximum. User requests will be sent to the controller and the following response will be distributed to the sub modules, which will update the view.
Core features (ROC-2)	The core features correlates with the core functionality in the controller. • RetrieveFiles = Show a user's files and folders • DownloadFile = Button to download a file to the user's hard drive • Delete = Button to delete a file • Etc.

Necessary features (ROC-3)	Features that are more likely to change according to the data: • File icons • Visual styling on the web page(css/html) • Menus • File path • etc. If the data did not show that this feature was essential for the product but still necessary in order to complete the current MVP it was places here.

Flopfile's early impressions on the LSM architecture

When every sub-modules had been carefully considered in order to allow future changes with only minimal impact, it was relevant to know what the members of Flopfile's impressions were before they actually saw the benefits of this. It was obvious that the new architecture was more time consuming than what the Flopfile team was used to.

"Building an architecture on the foundations that you aren't really sure of anything was something I haven't tried before. The notion of rate-of-change and separating these seems like a smart idea but I feel like it slows everything down because of the constant analyzing for future changes." (Johan, Appendix B4).

Regardless of the added time, the expectation was now that the new architecture could sustain possible changes so the team can focus on the LSM process.

Creating the second test plan

After having created their second MVP, Flopfile was ready to plan the second test. This test would be their first test of an actual working prototype, were the users would physically be able to click on different buttons in the application instead of just having to imagine what would happen in different cases. Therefore Flopfile chose to draw inspiration from the first test but structure this test a little different. Flopfile's testplan was structured in the following way:

- Locate 10 potential early adopters among university students (test persons)
    - Contact them face to face
    - Different test persons than from the first test due to variety in answers

    - The test persons must use at least one of the following cloud storages; Dropbox, Google Drive and/or OneDrive
- Perform the test
    - Give a description of the product
    - Introduce the test persons to the test
        - A list of difference "problems" the user should try to solve
        - Observe how the user manages the different tasks
        - Follow-up questions on the application and the user experience
    - End the test with two questions:
        - "Would you be interested in buying this product once it is finished?"
        - "Would you like to participate in future product tests?"
- Evaluate the results

Executing the test

The purpose of the second MVP test was different than the initial test. The people in the first test confirmed the problem of organizing files in cloud storages.

"The first test gave us a lot of valuable information, but the most important thing was the fact that people actually found this sort of tool useful." (Johan, Appendix B3)

As LSM states it is important at this stage not to develop the product for launch and hope for the best, you need to constantly validate whether you are on the right track or if you need to change things.

The test users were again students from the IT-University. However, now that they would be presented with an actual working prototype it was important to have people with different level of IT skills. If the test users consisted of 90 % software engineers, the test data would possibly be bias and not necessarily reflect the actual population. Therefore, various design and business students also participated in the test.

The interviewer started by giving a brief description of the application so the user was not completely in the dark before the test phase. The test phase consisted of various tasks they needed to do on the website. These tasks were all designed to test all of the core functionality and what a regular user would do on this website. The questions were phrased as problems they would have to solve using the tool, e.g., *"You cannot remember in which cloud storage you have stored your high* *school pictures".* This gave a more realistic setting for the test users instead of just giving them simple commands.

The most important results can be seen below:

Initial hypotheses	**Evidence showed**
In order to register a user on the website, a 'signup with Facebook' option is enough.	While all the participants did use Facebook only 70% were willing to actually register, as they were worried about their privacy.
People are willing to let the application manages their files.	30% of the participants needed some information about how their personal data would be processed.
People understand how actions on the website actually influence their cloud storage accounts.	There was a very good understanding here. All of the participants instinctively knew that Flopfile was directly connected to their own cloud storage.

After the program test some follow up question were asked. These include *"How was the general overview of your files."* and *"do you think this application will solve your problems with managing multiple cloud storages?"*.

The questions gave a more in depth look at what the users actually thought of the program, instead of just testing the interface. A very interesting finding was the fact that 70% of the participants asked why they could not send their files to their friends or colleagues. The overview of the files was ok, but it was not really a problem since the majority had different purposes for each cloud storage. Most of them knew that all their work documents were in one single cloud storages and their personal documents in another. However, if you could share these files without having to worry about which cloud storage the receiver uses and thereby avoid email attachments, it would really solve a problem for them. This was surprising for Flopfile since the initial test almost gave the opposite results by having focus on overview and cutting the sharing functionality.

"I think since they could now see their own files and not just a mockup it helped to set things in perspective. This meant that they could see that sharing was actually the biggest issue" (Kristian, Appendix B5)

In addition they discovered that the numbers for people wanting to buy the product had not changed much since the first test.

Pivot or preserver

Based on the new empirical data, Flopfile decided that simply persevering with the original strategy would not be optimal. Therefore, it was necessary to do a "customer need pivot" in order to shift the focus from file organization to file sharing. The overall vision would remain the same but the strategy needed to be changed.

"Looking at the data we simply cannot justify to keep going in the same direction. The tests showed that the real issue is the file sharing across cloud storages and that will require a complete change both in our business strategy but also the program." (Johan, Appendix B5)

The new strategy would now be how people can share files of any size regardless of cloud provider. This meant that you could:

- Eliminate email attachments (especially larger files)
- Eliminate the need to transfer files psychically via a flash drive.
- Eliminate the need to know which cloud provider a person has when transferring files.

This would require quite a big change in the MVP in order to allow this functionality.

5.2.3 Steer - second iteration

The third iteration builds on the results gained in the previous tests and tries to improve the MVP even further. Based on the results from last test the team had decided to make a change in their strategy and thus a pivot was necessary. This was especially interesting regarding the software since it would require certain changes in order to accommodate this and therefore a chance to see how the LSM architecture would handle it.

"We need to change how the users view the program. Instead of focusing on directory operations like rename, delete and add we should simply display the files and make them easy to share." (Johan, Appendix B6)

It was important to recognize that this would not require an entirely new product, but rather an alternate version of the existing one. The team discussed how the program should be changed in order to meet the new requirements. They separated the components into what would have to change and what could be left as it was.

These components could be left unchanged:

- The external data
- Communication between modules (ROC-1)
- Most core functionality and core features (ROC-2)
- A few necessary functionality and necessary features (ROC-3)

That meant that these needed to be changed:

- Big part of the internal data
- Some core functionality and core features (ROC-2)
- A lot of necessary functionality and necessary features (ROC-3)

"We definitely saw that many of the ROC-3 components needed to be changed whereas the ROC-1 components could ultimately be left alone" (Johan, Appendix B6)

Model

In order to implement the sharing functionality the program needed to store additional information. This information would contain; who have shared the file, which file was shared and to whom was it shared with. There was also some data in the internal database that was no longer necessary and needed to be removed.

"When it came to the data it was actually very easy to add and delete attribute in the internal database since it was completely separated from the external due to the 2 view models we added as part of ROC-1 in the Model." (Johan, Appendix B6)

Controller and View

The change in controller and view were more complicated than in the model. Some of the '*core functionality*' needed to be changed which would directly affect the *'necessary functionality'* module.

"The redesign of the components in ROC-2 both view and controller generally worked well. The fact that we knew these changes would only affect these modules and the lower modules allowed us to focus on the actual integration of the new features instead of having to redo a lot of working code." (Kristian, Appendix B7)

Core operations like 'd*elete'*, *'rename'* and *'create folder'* were removed and replaced with *'share file'* and *'download shared file'.* Like in the previous version, these ROC-2 operations would not know which specific cloud provider the file was located in.
Changing the ROC-3 components were easy and effective due to their lack of dependencies.

*"Everything in ROC-3 in both controller and view were very specific features and therefore we knew they were subject to change. This actually led to the change being extremely easy. Since these components could only depend on the module we could freely remove and add components without much rework in the program." (*Kristian*, Appendix B7)*

Even though changing requirements in ROC-3 was very easy for the Flopfile team, the core functionality and feature did create some problems.

"We did experience some issues when changing the ROC-2 components in both the controller and view. The fact that we did not have to worry about the communication was very nice. However sometimes communication was not very intuitive since we are not use to work like this. This did create some difficulties for us although it did not happen very often." (Kristian, Appendix B7)

Flopfile explained that when a certain button was changed in the view, it could be difficult to Figure out where the corresponding component in the controller was in order to make the button work as intended. Due to the way they had to implement the communication modules they could not rely on their knowledge from traditional methods. However, Flopfile did only experience this a few times.

Testing the new MVP

The purpose of the third MVP test was to illuminate how the potential users saw the new program and ultimately if the pivot was the right move. The test plan was very similar to the previous once with 10 users performing specific tasks.

Results

Initial hypotheses	**Evidence showed**
People could easily figure out how to share their files with a specific person.	80% completed to test without any problems, whereas 20% had trouble with the menu.
Some people wanted to signup via Facebook and some wanted to create a new account	70% chose to sign up via Facebook 30% created a new account and manually wrote their personal information

People though sharing files across cloud providers was something they could use in their everyday life.	80% of the people answered that they encountered this sort of problem before and liked how this product solved it.
An overview of all their files was still not as important as sharing.	90% of the participants answered that sharing files in cloud storages was a much bigger problem than an overview.

In addition, they saw an increase in potential buyers for the product. Now 40% answered "Yes" when asked whether or not they wanted to buy the product and 50% answered "Maybe".

Due to the very positive results, they decided to persevere with the new strategy. The sharing functionality was still very relevant to the users and a much bigger help that the previous file overview strategy.

5.3 Evaluating LSM and the guidelines

After having observed Flopfile for almost two months it was possible to evaluate how they had perceived the use of LSM including our guidelines. We wanted to discover if they had seen the LSM process as a success and with their current knowledge describe more in depth what their opinion was concerning the use of our guidelines during this process.

We chose to set up a final structured interview with the founders of Flopfile and asked them about their opinions on different parts of the process. They all agreed that the LSM process had been a great success and that they had learned a lot about how their customers think as well as the value of validated learning (Mads, Appendix A2). We structured our relevant findings based on each separate guideline.

5.3.1 Guideline 1

The first guidelines focused on keeping options open for as long as possible. Due to this guideline, Flopfile had refrained from doing any actual programming until they had collected enough empirical data to justify creating an actual functional MVP. However, after the positive results gained in the mockup test they knew that this type of product was actually something the customers wanted and therefore the implementation phase could begin.

*"It was actually quite difficult to delay any actually programming since you are so certain that this product is the greatest thing ever. But it has become clear to me that without any knowledge of what our potential customers wants we could easily be wasting our time developing features that was unnecessary." (*Kristian, Appendix B2).

As Kristian says, it was almost intuitive for the team to immediately start programming because they all felt this was the greatest idea ever, but following guideline 1 they chose to delay this decision and instead created the mockups. This allowed them to refrain from faith-based decisions and instead rely solely on the empirical data.

The general feeling about the guideline was positive. Johan mentioned that he believes the advise to create a non-technical prototype as the first MVP is great. Without this part the team would have started to develop a prototype using code, which would have been much more time consuming and with the danger of scrapping a lot of code later (Johan, Appendix A2).

"[...] I think this guideline is good and I will definitely continue to use it during our project since I believe it has helped us to minimize waste. If not for this guideline we would probably have scrapped more code during the past two months which really would demoralize me." (Johan, Appendix A2)

Even though the Flopfile team were satisfied with the guideline they did experience some negative consequences while using it. The fact that they had to keep many options open made the development process more time consuming. You have to consider various tasks that might become a part of the program and you cannot make certain features essential for the program unless you have extensive empirical research.

5.3.2 Guideline 2

The second guideline focused on building a sturdy foundation for the software in order to accommodate changing requirements. When building the architecture the developers of Flopfile recognized that, the LSM architecture builds on the classic MVC pattern, which did help a lot getting started.

*"We knew that it build on the classic MVC architecture which we as developers are very used to and also intended to use in the project in some way. But the extra layer and the notion of a rate of change was something we have not heard before." (*Kristian*, Appendix A2)*

The process of separating each component into different rate of change categories did create some struggles for Flopfile.

"It was not always so intuitive to know which components should be given what ROC" (Kristian, Appendix A2)

The problem was that even though the guideline said you should rely on empirical data, the fact that they only had conducted one test at this stage meant that their data was quite limited. This also meant that they had to spend relatively much time building the architecture.
Due to the nature of the LSM architecture the team was forced to constantly separate independent components and make sure that there were not any unnecessary dependencies that could impact the program in case of changing requirement.

After the pivot, Flopfile had to change some features in the software that ultimately did create some communication issues described in the second iteration of Steer. However, despite these issues, Flopfile did manage to successfully pivot with relatively minimal impact on the program and very little code scrapping.

*"I was extremely impressed with how the LSM architecture allowed to change the program without creating a domino effect of bug fixing. However, this level of separation might be a little too extreme since it did work against us at one point." (*Kristian*, Appendix A2)*

5.3.3 Guideline 3

The third guideline was met with a bit of skepticism for the members of Flopfile. They mentioned that they did not understand the necessity of this guideline since they already knew how to work together and what program they would build. Although this was the initial thought Johan describes how the guidelines helped him during the two months:

"For me it has been really helpful. Usually I am the one trying to develop Mads' ideas or investigate if they are even possible - he is normally the one generating the ideas for Flopfile. Now we are all part of it, even Kristian is enthusiastic about it. One of the downsides from my perspective is that now I spend too much time thinking about new cool features for our product every day and do not write as

many lines of code as I used to in a day, but I guess I'll just have to learn the balance." (Johan, Appendix B3)

As Johan describes it enhanced the opportunity for him and Kristian to be a valuable part of the idea generation process. Usually they would have focused more on the actual coding process hoping that Mads would generate some good ideas for their project. In addition, Mads describes how he has changed his mind during the project:

"[...] I must admit though that the guidelines have helped us during the past two months. I actually think we have become more innovative during the past two months using these guideline than we would have been without them." (Mads, Appendix A2)

According the Mads the team has become more innovative in generel. The amount of ideas they have generated using the third guideline was very impressive and certainly more than usual.

In the end we asked Flopfile whether or not they would continue to use LSM and our guidelines in the future. The founders told us that they would definitely continue to use LSM including our guidelines. The amount of learning has been much higher than they could anticipate (Mads, Appendix A2).

5.3.4 Summarizing our findings

Guideline	Theoretical expectations	Empirical findings
Guideline 1: Minimizing waste in your software	**1)** Changes in the software will not result in large code scrapping.	**1)** Avoiding locking in decisions early did allow Flopfile to hold off on features they did not have sufficient empirical data on. Some of these features later showed to be unnecessary and therefore would be considered waste. However, it was much harder and time consuming to develop software when you want all your options to be open. The decisions you make in the program have to consider all the possible changes that might occur which can be very resourceful (Johan, Appendix A2).
Guideline 2: Create a LSM architecture	**1)** Before any actual programming, the entrepreneur will spent time defining how each module should communicate. This platform for communication will not change very much during the startup process. **2)** Changing, adding or deleting specific features will be easy and only require minimal rework in other areas of the program.	**1)** The actual communication between the three modules remained constant even during the pivot. This allowed the Flopfile team to focus on the actual feature implementation. However, this was quite time consuming to implement. (Johan, Appendix B7) **2)** During the pivot the team had to change, add and delete certain features to fit with the new strategy. The impact on other parts of the programs were minimal and the changes were easy to implement following the architecture.
Guideline 3: Innovation as a development activity	**1)** Several innovative ideas for possible solutions/features etc. will be created due to each team members involvement in the idea generation process. This will give the team the ability to select their solution from multiple ideas and not just choose the first and cheapest solution. **2)** Each team member will end up having a good understanding of the product as a whole during the development phase including which parts are being created by the others.	**1)** By focusing on innovation using guideline 3, the team members at Flopfile all contributed to the idea generation process (Johan, Appendix B3). In addition Flopfile managed to create numerous ideas. Mads Larsson (Appendix A2) describes the number of ideas generated during the two months as more than usual. **2)** Flopfile's team members had a good understanding of the product as a whole due to their innovation sessions.

6) Discussion

In this section we have discussed our findings and reflected on how this can be considered a valid contribution to the researching universe. We have divided it into four sub sections including; wrapping up the findings, limitations, implications and future research.

6.1 Wrapping up the findings

Now that we have gathered our data from the case study with Flopfile it is time to reflect on our findings and the process leading to them. We will be discussing the LSM process as well as each guideline individually and reflect upon how the guidelines have been applied including how this has affected our results. We will focus on the comparison between our empirical findings and the theoretical expectations and discuss what we can derive from our research.

6.1.1 The LSM process

First, we will discuss how Flopfile has chosen to apply the LSM process. This is relevant since our guidelines are customized for use in cooperation with the LSM process and the results might therefore be affected if Flopfile has chosen to deviate from the LSM process or modify it.

Overall Flopfile has followed the LSM process well (according to the theory) although with some adjustments. Primarily Flopfile focused less on testing than the LSM advises. As described in the theory almost every aspect of LSM is connected to testing. We identified some areas where Flopfile deviated from the LSM. These areas are;

- They did not test their growth hypothesis (during the initial hypothesis testing)
- They did not validate their third hypothesis before creating their third MVP
- They did not set a clear baseline before each test.
- They did not use innovation accounting with specific goals before each test

By not following the LSM as advised by the theory it increases the risk of generating waste. This directly relates to guideline 1 which states that decisions on implementation should be made as late as possible to allow for sufficient empirical evidence. By neglecting proper testing in specific areas it increases the risk of making wrong assumptions about the world.

Although this type of deviation from the LSM process increases the risk of making wrong assumptions we would categorize this risk as minor. This is because the deviations only account for a small part of the whole LSM process. We would therefore argue that even though the risk increases, this increase will be minor and would thus not impact the validity of our results.

6.1.2 Guideline 1

In order to evaluate the results for guideline 1 (minimizing waste) it is important to discuss whether or not Flopfile has used this guideline as intended. In general, we can say that Flopfile has followed the guideline completely as intended during our research period. Flopfile has throughout the process kept options open for as long as they thought possible and avoided locking in decisions. They have also created a non-programming prototype to gather sufficient empirical data from potential customers during the beginning. This creates a good foundation for evaluating our findings.

Comparing our findings to the theoretical expectations we see that the guidelines has helped Flopfile minimizing waste when discussing potential code scrapping. According to Poppendieck (2003) a key principle in lean software development is the notion of deciding at late as possible. Because you work in these uncertain environment you need to build a capacity for change in the software by only making decisions when you have sufficient empirical data. With the avoidance of an early functional prototype and the fact that they never made important decisions prematurely Flopfile meant they only implemented features that was actually necessary for the user. This meant that a lot the potential waste in form of unnecessary features was avoided because of this guideline.

It should be recognized that during the pivot Flopfile did have to delete already implemented features. Based on the second test they discovered that their current strategy was not aligned with the user's actual needs. This meant that some working operations were no longer necessary and had to be scrapped. However, we would not categorize this as waste since theses features were necessary in order effectively evaluate the strategy and ultimately decide to pivot. Even though they did not end up in the current version they were still necessary for the process.

Although this is the case, this guideline might instead have created another form of waste - too much extra time being spent creating the software due to keeping options open. This prolongs the overall creation process since the project will need extra planning and verification before each decision are to be made. This could have the effect that the Flopfile team would simply not create the same amount of code although they would decrease the risk of this code are to be scrapped later on. However, we would argue that the benefits gained from this guidelines far exceeds the extra time invested.

What we can derive from our findings is that this guideline will help companies like Flopfile minimizing their amount of code scrapping since it prevents a number of situations where this would occur.

6.1.3 Guideline 2

One of the biggest technical challenges in LSM according to the literature was how it supposedly devalued architecture and because of this, the entrepreneur would have to scrap working code when requirements changed. The second guideline aimed on solving this problem with a customized software architecture called the LSM architecture.

We recognize that integrating a certain software architecture into your program can be done in many different ways. It depends on the choice of programming language and which specific framework one chooses to use. For this reason, we will only analyze if the overall approach and structure described in the LSM architecture matches the concrete implementation from Flopfile and discuss the effect it had. The key thing about the LSM architecture was how it introduced a completely new layer of separation in the code, which corresponded to how likely, the specific components were to change.
We saw that Flopfile chose to wait for the second iteration to build a functional MVP (because of the first guideline). This was important since the guideline stated that the rate-of-change classification of these components should always be based on actual empirical data and not just intuition. We saw how Flopfile rated each components based on how sure they were that this feature was a core part of the program or if it was likely to change.

The LSM architecture was based on several techniques describes by Coplion (2010) on how one can create an architecture that works in extremely uncertain environments. According to the theory, with the help of LSM architecture the program should be able to handle changing requirements without breaking the program, which really showed during the pivot that Flopfile experienced. The change in strategy required quite a lot of changes in every one of the three modules. Here we saw just how flexible the program was and how one could easily change certain features without breaking the overall structure.

Due to these arguments, we can conclude that Flopfile's implementation of the LSM architecture was in accordance with our expectation and it did work according to the theory. However, there was one issue that needs to be discussed. Flopfile mentioned in several interviews and observation that the amount of time you had to invest in the early phase to implement this architecture was quite considerable. Therefore, one could ask if the time investment on the LSM architecture is actually worth it to prevent possible code scrapping and being less resistance to change. Based on the data we have gathered we would definitely argue that it is worth the time investment. We saw during the pivot how easy it was to change the components with a high rate of change, which allowed the developers to focus on the actual implementation of the new features and not fixing a chain reaction of bugs in the code or in the worst case completely redoing the MVP. Even though you have to invest more time initially, because of the uncertainty in the environment the second guideline did result in both better quality and lower development time.
Therefore, we can derive that while the LSM architecture does require the entrepreneurs to invest some time in it initially, it will provide you with a sturdy foundation for the program that is optimized to work in extremely uncertain environments.

6.1.4 Guideline 3

As with the other guidelines, it is important to know if guideline 3 (improving innovation) was applied correctly in order to discuss its validity. When viewing the process that Flopfile used during our research and how they have applied guideline 3 in particular we can see that they have modified it a bit. The intention was to use this guideline during the idea-phase of the Build-Measure-Learn feedback loop and in the learn-phase. Flopfile has on the other hand, chosen to combine these two sessions and only use this guideline once every cycle in the idea-phase where they also included what they had learned from the previous cycle. By doing so and choosing not to separate these two sessions the team does not necessarily focus enough on the lean software development principle of Amplifying learning (Cobb, 2011).

The might result in a shift more towards new features and less towards improving existing. By doing so the team might neglect valuable contributions from the learn-phase although this might be difficult to prove based on the available data.

Comparing our findings to the theoretical expectations, we can see that not only did Flopfile create several innovative ideas for possible solutions but they actually created more than usual according to themselves. Included in the theoretical expectations is that each team member of Flopfile will end up having a good understanding of the project as a whole including what the others are working on. The results suggest that this expectation has proven to be true although this guideline did not directly contribute to the knowledge of what each team member was working on. This could be connected to the fact that Flopfile chose to add morning Scrum meetings to the LSM process, which have made this part superfluous during the innovation sessions.

What we can derive from our findings is that this guideline will build the foundation for companies like Flopfile to create numerous innovative ideas since it increases the focus on this specifically. It will also help with the general overview of the product for each team member. What we cannot conclude based on our findings is whether this guideline will increase awareness on what the other team members are working on.

6.2 Implications

With the concrete results thoroughly analyzed, it is relevant to discuss if and how these findings will actually influence both the startup environment and the field of research within this area. We draw inspiration from Shirley Gregor's taxonomy of theory types in information systems (Gregor, 2006) to classify the type of research and what constitutes as a contribution to knowledge with regards to our research. Like most design science research projects, our research can be classified as a theory for design and action. This is because the goal is to investigate how you can overcome the technical barriers within LSM and thus create a better foundation for entrepreneurs in the IT sector. The theory focuses on the principles of form, function and justificatory theoretical knowledge needed in order to create an artifact to solve these problems; this is the essence of a design and action theory. In order to evaluate the contribution to knowledge for such a theory the following criteria has to be discussed; utility to a community of users, the novelty of the artifact and the persuasiveness of claims that it is effective. Furthermore, things like completeness, simplicity, ease of use and "interestingness" are also very relevant to evaluate (Gregor, 2006).

In order to discuss the utility to the community of users it is necessary to define the overall goal of the thesis: We want to create a better foundation for success in IT startups by aligning the software development approach with the business methodology, in our case LSM. If the technical aspect has already been considered when starting a business you can prevent some of the technical challenges you often face in such an environment. The justificatory knowledge concerning this gap in the existing design theory was derived from different respected authors and entrepreneurs openly criticizing LSM for being too detached from modern software development practices. Because of this, we argue that the sheer usefulness of this theory is very high for the community. With the elimination of this barrier, the entrepreneurs will be much more likely to develop products that are sustainable even in the very uncertain surroundings you face in the startup environment. With the current rate of success being so low for these types of companies with 25% of new startups failing within one year (WSJ, 2012) and 75% of venture-backed startups not returning investors capital (Shane, 2010) the guidelines are certainly relevant for the community. With the help of future research it will be possible to transform it from a nascent design theory to a full design theory. We believe that we will see an increase in success rate from IT startups using the guidelines and thus an increase in the overall revenue generated from these types of companies. We recognize that with the scope and limitations of this thesis (described in the limitation section) the solid evidence for this claim can seem non-decisive. However, based on the evidence gathered from a naturalistic setting coupled with the amount of literature used to create the artifact, we argue that the evidence for the theory's utility to the community is convincing.

Another important topic that is necessary to discuss in order to evaluate the knowledge contribution of the research is the novelty of the artifact. This will require a deeper analysis of the 'research gap'. As mentioned in the introduction, the startup methodology 'lean startup' has received increasing attention in recent years. The book 'The Lean Startup' by Eric Ries (Ries, 2011) is the number 1 best-selling book on amazon in the category 'New Business Enterprises' (Amazon, 2015). Many new entrepreneurs has adapted this methodology into their own venture because of the revolutionary techniques it offers. However, there is naturally a lack of scientific research exploring this methodology since it is still relatively new. Even though a set of technical challenges within the LSM has been identified by different sources, there exist little to none research offering concrete solutions to these problems. Bridging this research gap is very relevant since it can cause unnecessary delays in the software development and inferior products according to the literature described in technical challenges section. We recognize that these challenges may not occur in every new company - some might face a completely other set of technical challenges, which are not described in the guidelines. The novelty of the artifact comes from how it puts focus on this research gap in the LSM and tries to offer concrete solutions in a way that is easily extendable.

A discussion on the artifacts mutability is also very relevant in this context. Gregor & Jones (2007) describes artifact mutability as *"the changes in state of the artifact anticipated in the theory"* (Gregor

& Jones, 2007, p.322). The technical guidelines are designed to be interdependent with the LSM process. The literature used to create these guidelines are all based around working in very uncertain environments like one faces in a new startup. Because of this, we argue that the degree of mutability of our artifact is low since it builds on many different theories most people in the industry are familiar with like the architectural pattern *'MVC'* and the agile methodology '*Scrum'*. The artifact does not try to introduce a set of revolutionary new techniques that will disrupt the whole software environment and change the way we think about technology. Instead, it encourages people to keep using these popular techniques but just in a modified version that is specifically designed to the startup environment.

Concerning the *'completeness'* of the study, which is another key factor in the evaluation of the knowledge contribution there are some interesting aspects that should be discussed. First, it would be naive to strive for a complete elimination of every technical problem encountered while using the LSM - that is simply not possible due to the uncertain environment and the fact that every company is different. Therefore, the scope of the thesis was to focus on some the known problems and create guidelines that will help entrepreneurs to prevent these problems. Therefore, we argue that the completeness is limited to the available knowledge of the LSM we have today. There are a number of other factors that influence the completeness of study such as the restricted data gathered, the limited time frame and the type output of the thesis. This is discussed in the limitation section.

Another implication that is not directly mentioned by Shirley Gregor (2006) is how relevant our research will be to other kinds of startups. Even though the focus has always been on companies where the main product is a piece of software it is interesting to discuss what will happen if we eliminate this requirement and look at startups companies in general. It is of cause obvious that the specific challenges and guidelines will not be particularly useful due to its very technical nature. However, the idea of aligning the LSM process with the production process is very important regardless of the type of product. We have discovered how important it is to create synergies between your business decisions and how you actually develop the product. Further research can use the process of this thesis as a foundation for creating a new set of guidelines designed for other areas in the startup environment such as manufacturing.

The implications of the study has now been discussed using the different criteria described by Shirley Gregor in a design and action study (Gregor, 2006). We have argued that such an artifact that tries to bridge the research gap between LSM and the software process could help increase the success rate in startups and ultimately create a more competitive marketplace.

6.3 Limitations

With the implication carefully considered it is important to recognize that this research does have certain limitations that needs to be addressed.

One of the biggest limitations in the thesis comes from the restricted empirical data gathered. Due to limited resources and the scope of the thesis, it was only possible to evaluate our guidelines in one company. Even though the research within this company was quite extensive, the sheer evidence of the actual effect of the guidelines could be criticized for being too narrow to conclude anything substantial. If the research had a range of different companies, which all utilized the guidelines in their startup process it would help to justify whether or not they actually solves the technical challenges that were identified.

Another limitation comes from the fact that you do not have evidence that the company will indeed blossom and grow as time progresses since this was not possible within the given time frame. We did see how our guidelines helped the software adapt to changing requirement and therefore prevent many of the technical challenges you face in LSM, but if it does not correlate with the success rate of software-startups it is not very relevant in this context.

However, even though we have acknowledged these limitations we still think that the project is relevant and will indeed be a valuable contribution. This stems from the fact that we were well aware of these limitation even before we began the evaluation of our guidelines. Because of this, we chose a methodology with an iterative approach that allowed further research to be built directly on our results. You often use the results in design science to redefine the problem and conduct further evaluations based on that.
Another argument is the type of contribution that will be the output of the thesis. As mentioned in the methodology, the output of the thesis would not be a full design theory; this would require much more research with different companies using the guidelines and a much higher maturity level of evidence. Instead, the output is a nascent design theory, which includes a defined artifact that aims to solve some specific problems but is not yet mature enough to be considered a fully proven design theory. In our case, this artifact is our technical guidelines that will help solve some of the technical challenges within LSM such as devaluing architecture, minimizing waste and lack of innovative ideas. With our extensive interviews and observation sessions, we do have solid evidence that these guideline did indeed work for our case company Flopfile. We know that one startup

company does not represent the entire community and just because it created positive results in their case does not mean it necessarily works on similar companies, especially within the startup environment with its extreme uncertainty. However, we did see how specifically the guidelines affected the process and how the people used them in different situations. Based on the results gained in the evaluation we would argue that this is a relevant contribution even though there are different limitations.

6.4 Future research

In addition to the limitation section we will elaborate on how our research can be improved/extended in the future. The reason why it would be particularly relevant to do additional research within this field would be that this thesis strive to create a nascent design theory (or a 'level 2 contribution' (Hevner & Gregor, 2013) as mentioned in the methodology section). In order to generalize some of the findings from this thesis and thereby create a full design theory, additional research is needed. We have therefore created a list of possible research subjects that might be relevant for future research.

6.4.1 Additional IT-startups

The results of this thesis are based on a single IT-startup company and it is therefore very difficult to discuss whether or not other IT-startups will experience the same results if they were to apply our guidelines. Analyzing an additional number of IT-startups will definitely help improving the generalizability of our results.

6.4.2 Other startups

This thesis centers around creating guidelines for IT-companies with software development as one of their primary activities. Although this is the case, other companies might also benefit from our findings. More specifically companies with software development as a secondary activity (e.g. web shops, etc.) could be an interesting group to do future research with in order to generalize our findings across different lines of business.

6.4.3 New projects in established companies

As described in the theory section when elaborating on LSM, it was mentioned that LSM was not just a methodology for entrepreneurs, but could just as well be used by '*intrepreneurs*' which is entrepreneurs inside a well established company with the responsibility of launching a brand new product with high uncertainty. Therefore if LSM works for well established companies our results might as well be interesting to try to replicate on projects with software development as a primary activity.

6.4.4 Longer research period

Extending the research period for each case company will provide data describing how our guidelines work in the long run. Since we have only spent two months with our case company, we have no data describing the long term effects and whether or not they can actually help increasing the success rate of IT-startups.

6.4.5 Additional guidelines

In general, the overall purpose of our guidelines are to improve the success rate of IT-startups. We have developed a set of three guidelines to prevent three specific problems and thereby improve this success rate. Even though, combined with additional research, our guidelines might turn out to actually improve this success rate they do not solve every problem an IT-startup faces. Therefore by doing additional research around other challenges IT-startups might face, other guidelines could be created in addition to ours and improve the success rate even more.

7) Conclusion

The purpose of this thesis was to explore how to create a better foundation for IT startups and improve their success rate. Recently, a new and controversial approach has emerges on how startups should be managed in order to improve this low success – known as the lean startup methodology (LSM). Even though LSM has received critical acclaim it does present different software related challenges when applied in an IT-startup. Thus the following research question was defined:

"How can you create preventative guidelines for the technical challenges one faces in the Lean Startup Methodology?"

A set of technical challenges were identified based on available literature concerning the LSM. In order to create preventative guidelines for this set of challenges additional literature was used. Specifically we analyzed literature that deals with software development in very uncertain environments like LSM. This includes the choice of software development methodology, the software architectures and the decision making process. With the help of this literature it was possible to established guidelines which will have a preventative effect on the identified challenges.

It resulted in the creation of three specific guidelines customized for LSM and agile software development methodologies like Scrum or Kanban:

1. Minimizing waste in your software
2. Create a LSM architecture
3. Innovation as a development activity

These three guidelines were evaluated in a two months case study with the startup company Flopfile. The evaluation process illuminated a number of interesting findings on the guidelines when applied to the LSM process and the agile development environment.

The first guideline focused on preventing waste in a software process caused by decisions made prematurely without enough empirical evidence to justify them. This guideline achieved this by prolonging decisions for as long as possible and thereby keeping options open until the entrepreneur had gathered sufficient empirical evidence. From our case study we discovered that a number of unnecessary features were never implemented as a direct result of our guideline. These features would otherwise have been considered waste since they were not relevant in order to solve the users' problems.

The second guideline was directed towards a lack of attention to software architecture in startups using the LSM. Based on various literature about architecture in uncertain environment it was possible to create a customized LSM architecture that is specifically designed for new startup ventures who are using the LSM. It was discovered that by introducing an additional layer of separation in the architecture according to how likely a components are to change, the software becomes more much capable of handling changing requirements. By building the architecture based on empirical data and accepting that changes are bound to happen you build a capacity for change that can handle the extremely uncertain environment. This allowed the developers to focus on the actual implementation of new features and not fixing a chain reaction of bugs in the code because modifications were needed.

The third guideline tried to solve the lack of focus on innovation for startups using LSM and agile software development together. To do this a specific activity was created which states the need to actively incorporate innovation sessions to each iteration cycle in the LSM process during the development period. By incorporating this guideline in the environment of Flopfile we discovered that numerous ideas for possible solutions/features were created. This provided the team with multiple options for deciding in which direction their application should turn and how the end product should look like.

The contribution to knowledge for this thesis comes from how it bridges the research gap from LSM and modern software development practices. Currently, there exist little to none research offering concrete solutions to the technical challenges associated with LSM. These challenges can create longer development time and inferior products, which is why it is necessary to create preventive measures in order to create a better foundation for these startups.
It should be recognized that these finding are based on a restricted amount of empirical data since they were only collected from a single company. Therefore, the output of thesis can not be classified as a full proven design theory but rather a nascent one. However, with the help future research the maturity of the findings can grow as to create a full design theory.

We can therefore conclude that a set of guidelines has been created to prevent the technical challenges IT-startups face when applying LSM. These guidelines have been evaluated in the startup company Flopfile and has provided valuable results on its actual effect in a real life setting. With these findings we are now one step closer to aligning LSM with the software development process thus increasing the overall chances for success in IT-startups.

References:

Abbott, Martin L. & Fisher, Michael T. (2009) *“The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise”*, ISBN-13: 978-0137030422

Amazon (2015), http://www.amazon.com/Lean-Startup-Entrepreneurs-Continuous-Innovation/dp/0307887898/ref=sr_1_1?ie=UTF8&qid=1424857240&sr=8-1&keywords=the+lean+startup (Last viewed: 12/09 – 2015)

Ambler, Scott (2010) *“Agility@Scale: Strategies for Scaling Agile Software Development”*, IBM developerWorks, https://www.ibm.com/developerworks/community/blogs/ambler/entry/principles_lean_software_development?lang=en (Last viewed: 12/09 – 2015)

Bass, Len & Clements, Paul & Kazman, Rick (2003) *"Software Architecture in Practice"* Chapter 2.4 *"Why Is Software Architecture Important?"* Available online http://www.ece.ubc.ca/~matei/EECE417/BASS/ch02lev1sec4.html

Bernard, H.R. (2011) *“Research Methods in Anthropology”* 5th edition, AltaMira Press

Berry, Tim (2012) *“Lean Startups Need Business Plans, Too”*, Entrepreneur.com: http://www.entrepreneur.com/article/223878 (Last viewed: 12/09 – 2015)

Bjørnvig, G. & Coplien, J. (2010) *“Lean Architecture: for Agile Software Development”*, Hoboken: Wiley

Blank, Steve (2006) *"The Four Steps to the Epiphany"*, ISBN-13: 978-0989200509

Blomberg, J., & Giacomi, J. (1993) *“Ethnographic Field Methods and their Relation to Design”*, Participatory Design: Principles and Practices, ISBN-13: 978-0805809510

Boehm, Barry & Turner, Richard (2003) *“Balancing Agility and Discipline: A Guide for the Perplexed”*, ISBN-13: 978-0321186126, 1st edition

Bosch, Jan & Holmström, Helena & Björk Jens & Ljungblad, Jens (2013) *”The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups”*, ISBN-13: 978-3-642-44929-1

Brown, Millward (2014) *"Value of BrandZ™ Top 100 up 98 Percent Since 2006",* http://www.millwardbrown.com/mb-global/brand-strategy/brand-equity/brandz/top-global-brands/overview#sthash.1iLzDGDy.dpuf (Last viewed: 12/09 – 2015)

Burbeck, Steve (1992) *"How to use Model-View-Controller (MVC)"*, Smalltalk-80

Churchill, Gilbert (2009) *"Marketing Research: Methodological Foundations (with Qualtrics Card)"*, ISBN-13: 978-1439081013

Cobb, Charles G. (2011) *"Making Sense of Agile Project Management: Balancing Control and Agility"*, ISBN-13: 978-0470943366

Cohn, Mike (2014) *"My Primary Criticism of Scrum"*, http://www.mountaingoatsoftware.com/blog/my-primary-criticism-of-scrum (Last viewed: 12/09 – 2015)

Cooper, Brant & Vlaskovits, Patrick (2010) *"The Entrepreneur's Guide to Customer Development"*, ISBN-13: 978-0982743607

Coplion, James & Bjørnvig, Gertrud (2010) *"Lean Architecture for Agile Software Development"*, ISBN-13: 978-047068420

Crabtree, Benjamin F. & Miller, William L. (1999) *"Doing Qualitative Research (Research Methods for Primary Care)"*, ISBN-13: 978-0761914983

Deacon, John (1995) *"Model-View-Controller (MVC) Architecture"*

Engel, Rafael J. & Schutt, Russell K. (2009) *"Fundamentals of Social Work Research"*, ISBN-13: 978-1412954167

European Commission (EC) (2013) *"Entrepreneurship as a main driver for economic growth"* http://europa.eu/rapid/press-release_MEMO-13-5_en.htm (Last viewed: 12/09 – 2015)

Goldkuhl, Göran (2012) *"Pragmatism vs interpretivism in qualitative information systems research"*, European Journal of Information Systems, (21)

Gregor, Shirley (2006) *"The nature of theory in information systems"*, MIS Quarterly, Vol. 30, No. 3

Gregor, Shirley & Hevner, Alan R (2013) *"Positioning and presenting design science research for maximum impact"* MIS Quaterly, Vol. 37, No. 2

Gregor, Shirley & Jones, David (2007) *"The Anatomy of a Design Theory"* Vol 8, Issue 5, Article 2

Harper, David A. (1999) *"HOW ENTREPRENEURS LEARN: A POPPERIAN APPROACH AND ITS LIMITATIONS"* Working Paper Series, Department of Industrial Economics & Strategy, Copenhagen Business School No 99-3

Hevner, AR (2004) *"Design Science Research in Information Systems"*, MIS Quarterly, Vol. 28, No. 1

Holden, Mary T. & Lynch, Patrick (2004) *"Choosing the Appropriate Methodology: Understanding Research Philosophy"*, The Marketing Review, Vol. 4, No. 4

Hornaday, John A. & Aboud, John (1971) *"Characteristics of Successful Entrepreneurs"*, Personnel Psychology, Vol. 24, Issue 2

Hair, Joseph F. Jr. & Wolfinbarger, Mary & Money, Arthur H. & Samouel Phillip & Page, Michael J (2011) *"Essentials of Business Research Methods"*, ISBN-13: 978-0765626318

Jstanto (2008) *"The Customer Development Methodology"* Retrieved Sep 12, 2015 from slideshare.net: http://www.slideshare.net/jstanto/customer-development-mythology-by-steve-blank

Jussi Pasanen *"Minimum Viable Product"* Retrieved Sep 12, 2015 from twitter.com https://twitter.com/jopas/status/515301088660959233

Kniberg, H. (2007) *"Scrum and XP from the Trenches"*, 4Media Inc., InfoQ.com

Kvale, S. & Brinkmann, S. (2009) *"Interview - Introduktion til et håndværk"*, 2. edition, 3. oplag, Copenhagen: Hans Reitzels Forlag.

Leankit (2015) *"WHY USE KANBAN BOARDS?",* Retrieved Sep 12, 2015 from leankit.com http://leankit.com/kanban/why-use-kanban-boards/

Martyn Shuttleworth (2008) *"Falsifiability"*, Retrieved Sep 12, 2015 from Explorable.com: https://explorable.com/falsifiability

Mcclelland, David C. (1987) *"Characteristics of Successful Entrepreneurs"*, The Journal of Creative Behavior, Vol. 21, Issue 3

Newman, Isadore & Benz, Carolyn R. (1998) *"Qualitative-quantitative research methodology: exploring the interactive continuum"*, ISBN-13: 978-0809321506

Nobel, Carmen (2011) *"Teaching a 'Lean Startup' Strategy"*, http://hbswk.hbs.edu/item/6659.html (Last viewed: 12/09 – 2015)

Normalm, Thomas (2011) *"Achieving Lean Software Development"*, *http://www.diva-portal.org/smash/get/diva2:400627/fulltext01.pdf* (Last viewed: 12/09 – 2015)

Poppendieck, Mary & Poppendieck, Tom (2003), *"Lean Software Development: An Agile Toolkit"*, ISBN-13: 978-0321150783

Reenskaug, Trygve M. H. (2003) *"The Model-View-Controller (MVC) Its Past and Present"*, University of Oslo

Ries, Eric (2008) *"What is Customer Development"*, http://www.startuplessonslearned.com/2008/11/what-is-customer-development.html (Last viewed: 12/09 – 2015)

Ries, Eric (2009) *"Pivot, don't jump to a new vision"* http://www.startuplessonslearned.com/2009/06/pivot-dont-jump-to-new-vision.html (Last viewed: 12/09 – 2015)

Ries, Eric (2011) *"The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses"*, ISBN-13: 978-0307887894

Schwaber, Ken (2001) *"Agile Software Development with Scrum (Series in Agile Software Development)"*, ISBN-13: 978-0130676344, 1st edition

Schwaber, Ken (2010), *"Waterfall, Lean/Kanban, and Scrum"* Retrieved Jun 24, 2015 from kenschwaber.wordpress.com https://kenschwaber.wordpress.com/2010/06/10/waterfall-leankanban-and-scrum-2/

Shane, Scott A. (2010) *"The Illusions of Entrepreneurship: The Costly Myths That Entrepreneurs, Investors, and Policy Makers Live By"*, ISBN-13: 978-0300158564

Sharkey, Michael (2013) *"6 things wrong with the 'Lean Startup' model (and what to do about it)"*, http://venturebeat.com/2013/10/16/lean-startups-boo/ (Last viewed 12/09-2015)

Statista (2014) *"Leading motor vehicle manufacturers worldwide in 2014, based on global sales (in million units)"*, http://www.statista.com/statistics/275520/ranking-of-car-manufacturers-based-on-global-sales/

The Guardian (2009) "YouTube: the People's University of the Internet", http://www.theguardian.com/technology/2009/dec/02/youtube-peoples-university-internet (Last viewed 12/09-2015)

Vaishnavi, Vijay & Kuechler, Bill (2012) *"Design Science Research in Information Systems"*, http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf

Venable, John & Pries-Heje, Jan & Baskerville, Richard (2014) *"FEDS: a Framework for Evaluation in Design Science Research"*, http://www.palgrave-journals.com/ejis/journal/vaop/ncurrent/full/ejis201436a.html#aff1 (Last viewed 12/09-2015)

Wall Street Journal (WSJ) (2012) *"The Venture Capital Secret: 3 Out of 4 Start-Ups Fail"* http://www.wsj.com/articles/SB10000872396390443720204578004980476429190#articleTabs_comments%3D%26articleTabs%3Darticle (Last viewed 12/09-2015)

Wellington, J (2001) *"Educational Research: Contemporary Issues and Practical Approaches"*, ISBN-13: 978-0826449719

Wikipedia (MVC) (2015) *"Model–view–controller"* Retrieved Sep 12, 2015 from wikipedia.org https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Wikipedia (Scrum) (2015) "Scrum (software_development)" Retrieved Sep 12, 2015 from wikipedia.org https://en.wikipedia.org/wiki/Scrum_(software_development)

Wilson, J. (2010) *"Essentials of Business Research: A Guide to Doing Your Research Project"*, ISBN-13: 978-1848601338

Yin, R.K. (2008) *"Case Study Research: Design and Methods (Applied Social Research Methods)"*, 4th edition, ISBN-13: 978-1412960991

Appendix