Aalto University
School of Business

The Hidden Pitfalls of Kanban in Software Development

Information Systems Science

Master's thesis

Satu Anttila

2014

Department of Information and Service Economy
Aalto University
School of Business

Aalto University
School of Business

THE HIDDEN PITFALLS OF KANBAN IN SOFTWARE DEVELOPMENT

Master's thesis

Satu Anttila

20.11. 2014

Information and Service Management

Approved in the Department of Information and Service Economy

__ / __20__ and awarded the grade

_____

AALTO UNIVERSITY SCHOOL OF BUSINESS
Department of Information and Service Economy
Master's Thesis
Satu Anttila

ABSTRACT

Objectives of the Study

The objective of this thesis is to explore the hidden pitfalls of Kanban in software development. The aim is to discover the reasons behind the pitfalls and actions that nourish Kanban's failure.

Academic background and methodology

Software development industry has been shifting towards more iterative, responsive and people-oriented development methods, which present the values of lean and agile thinking. Being characterized as the antidote for bureaucracy, the shift towards agile development methodologies has been one of the most significant factors affecting the software industry. Kanban as an agile change management framework has been perceived as the fairy godmother of software development making the reasons behind struggling Kanban projects particularly interesting. Thesis interviews five agile experts in software development and studies their experiences of unsuccessful Kanban implementations. Emphasis is given to similar experiences and perceptions on how Kanban fails to redeem its value proposition.

Findings

The primary finding of the study is that an organization that is unable to change its mindset to support Kanban is a pitfall affecting the whole project, inevitably causing Kanban to fail. This is a challenge that all agile methods have in common. Agile can flourish only when agile values are being appreciated. The secondary finding is that many software teams claiming to be using Kanban have implemented only a shallow imitation of the real method thus creating a superficial implementation, which prevents achieving Kanban induced benefits. Furthermore, the erroneous human interpretation of what Kanban really is and how to apply it correctly is a key factor contributing to the emerge of the pitfalls.

Keywords

Kanban, software development, pitfall, agile, change management, framework

AALTO-YLIOPISTON KAUPPAKORKEAKOULU
Tieto- ja palvelutalouden laitos
Pro Gradu-tutkielma
Satu Anttila

20.11.2014

ABSTRAKTI

Tutkimuksen tavoitteet

Tutkimuksen tarkoitus oli selvittää ohjelmistokehityksessä käytetyn Kanban metodin piileviä sudenkuoppia. Tavoitteena oli ymmärtää miksi ja miten sudenkuopat saavat alkunsa, ja mitkä teot edistävät niiden syntyä.

Kirjallisuus ja metodologia

Ohjelmistokehitys on muuttumassa yhä iteratiivisemmaksi, ihmiskeskeisemmäksi ja herkemmin muutokseen reagoivaksi. Nämä arvot edustavat niin kutsuttua lean-ajattelutapaa ja ketterää ohjelmistokehitystä. Näitä kevyempiä metodeita kutsutaan myös byrokratian vasta-aineeksi, jotka muuttavat perusteellisesti ajattelutapaa tehdä tietojärjestelmiä. Kanban on ketterä viitekehys, jota on pidetty ohjelmistokehityksen hyvänä haltiattarena, mikä tekee epäonnistuneiden Kanban projektien taustoista erityisen mielenkiintoisia. Tutkimuksessa haastateltiin viittä ketterän ohjelmistokehityksen asiantuntijaa ja analysoitiin heidän kokemuksiaan epäonnistuneiden Kanban projekteista. Tutkimus antaa erityistä painoarvoa asiantuntijoiden yhteisille kokemuksille ja mielipiteille syistä, jotka estävät Kanbania lunastamasta arvolupaustaan.

Tulokset

Tutkimuksen merkittävin löydös oli, että kykenemättömyys ajattelutavan muuttamiseen ketterää ohjelmistokehitystä tukevaksi vaikuttaa koko projektiin ja johtaa lopulta väistämättä projektin epäonnistumiseen. Ketterät menetelmät voivat onnistua vain, jos ketteriä arvoja arvostetaan. Tutkimuksen toiseksi merkittävin löydös oli, että useat organisaatiot väittävät käyttävänsä Kanbania, kun kyseessä on oikeastaan vain pinnallisen jäljitelmä oikeasta Kanban metodista. Syyt sudenkuoppien takana vaihtelevat, mutta tutkimus tunnisti avaintekijänä organisaatioiden vääristyneen tulkinnan siitä, mikä Kanban oikeasti on ja miten implementoida Kanban osaksi nykyistä prosessia.

Avainsanat

Kanban, ohjelmistokehitys, sudenkuoppa, ketterä ohjelmistokehitys, muutosjohtaminen, viitekehys

Table of Content

List of Tables and Figures

ACKNOWLEDGEMENTS

I want to thank the following people for helping me write the thesis.

Sami Honkonen, for introducing me to Kanban in the first place.

Michael Holler, for helping me ask the right questions.

Sami Lilja, for knowing everything about everything.

Panu Liira, for pointing me in the right direction.

Toni Strandell, for making Kanban more graspable.

Juha Vakkila, for enlightening me with a totally different perspective.

Siru Kallström, for helping me express what I was thinking.

Matti Rossi, for constructive feedback along the way.

1 INTRODUCTION

The purpose of this chapter is to provide a background for the thesis and reveal the research gap. Hence, state the primary and secondary research question. The chapter also presents the structure of the thesis and provides a list of key definitions of the industry specific vocabulary.

1.1 BACKGROUND

A blossoming new software methodology known as the agile methods has emerged during the past two decades. Being characterized as the antidote for bureaucracy, the shift towards agile development methodologies has been one of the most significant factors affecting the software industry (Fowler, 2005; Leffingwell & Reinertsen, 2011). The high failure rate of agile methods' predecessor known as the Waterfall Model has caused the urge to provide better safety and governance for software projects. The number of failed Waterfall Model projects has been evaluated as high as 70 percent (Jones, 1999). According to the CHAOS report conducted by the Standish Group in 2012 software applications developed with agile methods have experienced three times the success rate compared to Waterfall method and have a much lower percentage of time and cost overruns. The report perceives the agile process as the universal remedy for software development project failure (Cohn, 2012). In addition, Standish Group implicates that 31% of the Waterfall projects are cancelled and that 53% of the projects will cost more than 189% of their estimates. The same study states that only 16% of Waterfall projects were completed as planned and as budgeted. Moreover, "successful" projects done by big companies had only 42% of the original features and functions that were defined before the start of the project (Standish Group, 1994).

The shift to more rapid, iterative and lightweight processes has been a consistent theme over time, delivering "outstanding benefits" on morale, quality and productivity. Ultimately, the lighter weight methods are creating a paradigm shift in the field of software development. According to survey made by Forrester Research in 2010 agile was the main methodology that most closely reflected the development process of 35 percent of Information Technology organizations (West et al., 2010). Originally the change initiative rose within individual teams that found themselves adopting agile methods such as XP, Scrum, and later on, Kanban

(Fowler, 2005). Gone is the traditional approach with rigorous requirement specifications and along with them the implied commitment to deliver a predefined product on a fixed schedule and with fixed resources. As a result, stability and predictability, which used to be the premises of software development, can no longer be taken for granted (Boehm 2006; Cleland-Huang 2013).

Studies have shown that around 80% of failed projects are related to incorrect requirements. This supports the use of Kanban, which allows changes to requirements during the process. Kanban takes a flexible, interactive and temporal approach to requirement management supporting the "just in time" thinking. As an adaptive approach to requirement management Kanban can lower resistance and accelerate capability improvement. It therefore seems likely that as time passes, Kanban will be seeing an increase in its usage in the software development community (Leffingwell & Reinertsen, 2011).

Kanban was originally created for manufacturing purposes. However, recently there has been a strong practitioner-driven movement endorsing the use of Kanban in software engineering. Due to Kanban's successful background in manufacturing the outcomes of applying Kanban to software development are expected to be high. The common perception of the new agile framework is insanely positive as though it is the fairy godmother of software development (Sprott, 2014). Kanban gets a standing ovation whenever it gets mentioned in a meeting. Even though Kanban experiences positive associations no theory is bulletproof. Every method has its weaknesses.

To conclude, Appelo (2010) presents a twisted scenario of how the failure rate of Kanban is going to be estimated as high as 75% - the same percentage skeptics claim another agile methodology Scrum to have. Appelo construes that the only explanation for the possible failure rate of Scrum is 75% is when the "ScrumButs" are included. ScrumButs are applications claiming to be Scrum but in present only a shallow illustration of the real method. Appelo believes that Kanban is likely to suffer from the same misapprehension, which is held in mind while searching for Kanban's hidden pitfalls in chapter four.

1.2 RESEARCH GAP

Although Kanban as a software development framework has gained momentum during the past few years, there has been next to no research about the potential pitfalls of Kanban nor about the difficulties the relatively new agile method faces. The few studies that have explored the dynamics of Kanban tend to concentrate purely on the benefits that the new agile method induces (Ikonen et al., 2011). Only few studies have touched on the reasons behind struggling agile processes (Cohn, 2008; Chapman & Powell, 2014; Tanner & von Willingh, 2014). Moreover, the studies discussing the reasons behind agile failures tend to be subjective in nature and based purely on own experience instead of on scientific data covering all agile methodologies and not just Kanban.

Because Kanban is still at the start of its adoption curve, the pitfalls of Kanban have remained hidden, lacking completely on empirical research. This serves as a source of motivation for this study, providing me with a green field approach. At that, the next section provides the research questions.

1.3 RESEARCH QUESTION

In accordance with the revealed research gap, this study explores the hidden pitfalls of Kanban driven software development. The purpose is to find answers relevant to the primary research question:

What are the hidden pitfalls of Kanban in software development?

The secondary research question supports the primary question by specifying the cause:

What are the reasons behind the pitfalls?

To clarify, the aim of the study is not to make statistical generalizations of the retrieved data. Instead, the aim is to shed light on a phenomenon by discussing similar experiences among industry the experts.

1.4 Research Structure

In pursuance of logical readability, this thesis is divided into seven chapters. Introduction aims to generate an overview of the software industry and justify the need of agile approach. Following, the next chapter presents relevant academic literature by reviewing the history of software development.

The third chapter introduces Kanban as a software development framework by encapsulating the essentials of the theory and practice. Thus, revealing the differences of Kanban in contrast to previous methods.

The fourth chapter explains the chosen research strategy and data collection methods. Moreover, the fourth chapter ascertains that the scientific research respected the values of validity and ethics. At the same time, the chapter forms an outline of the empirical research by introducing the qualitative approach, the interview method and the informants.

The fifth chapter along with chapter six analyses the data and summarizes the findings. The findings relevant to the research question are categorized in a table in the beginning of chapter six, which reflects them on previous research of agile hardships.

Finally, the seventh chapter discusses the research and presents the conclusions. The chapter also reflects the findings on previous agile research. Moreover, the chapter assesses the limitations of the study and offers suggestions for further research. The thesis is concluded with my personal reflections.

1.5 Definition of the Key Terms

Software project management

Software project management can be described as the art and science of planning and leading software projects. Planning includes budgeting, scheduling and defining the scope of the project.[1]

[1] Technopedia.com (Retrieved 20 October 2014)

Software development process

A software development process is a general term used to describe the over-arching process of developing software. The process can be used for the implementation of a single feature or for creating a whole new software system. While there is no standard definition, as we will see later on, most processes include similar activities:

- Requirement gathering
- Design
- Developing (coding)
- Testing
- Maintenance

Companies choose a development process from a range of methodologies to fit their customer demand and standards.[2]

Customer in software development context

The term customer in a software development context usually refers to a person or a party who has ordered the software feature/product (Honkonen, 2014a).

Product Backlog

The product backlog is an ordered list of requirements that is maintained for a product. It consists of features, bug fixes, customer requirements and so on: basically of everything that has to be done in order to successfully deliver a viable product/software to the customer (Cohn, 2014; Anderson, 2010).

Product Owner

The Product Owner (PO) is a person who represents stakeholders and acts as the voice of the customer. The PO is responsible that the software is valuable to business. The PO, together with the software development team, creates the product backlog. Commonly, the PO also prioritizes the tasks (Cohn, 2014). Note: different methodologies define the role of PO differently. The description above is a generalization of what PO usually refers to.

[2] Technopedia.com (Retrieved 20 October 2014)

Task

A task is an activity that needs to be accomplished in order to create the target product, the goal of the overall project. In software development context a task usually refers to a software feature or for example a defect in the code that needs to be corrected. The word “task” works as a synonym for an item, work task or user story (Anderson, 2010).

Efficiency

Efficiency is about doing tasks in an optimal way: maximizing efficiency and cost but not at the expense of quality. Efficiency is about optimizing output (Honkonen, 2014a).

Effectiveness

Effectiveness is about doing the right task - about completing activities and achieving goals.

The difference between efficient and effective is that efficiency tells how well one does something whereas effectiveness tells how useful it is (Honkonen, 2014a).

Lead-time

Lead-time is the total time it takes for a task to go through the process. From the first phase to the last phase, measured from the moment the task was created to the moment the task is completed (Rook, 2010).

Silver bullet & fairy godmother

Silver bullet is a term launched by Frederick Brooks (1982), which is known for the following argument: there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity. The term is commonly used for situation where one wants to emphasize the fact that no method has the power by itself to transform the process into a goldmine. Similarly, the term fairy godmother is also combined with self-evident benefits that just automatically appear with the new agile methodologies (Sprott, 2014).

2 HISTORY OF SOFTWARE DEVELOPMENT

This chapter presents the young field of software development starting with the traditional Waterfall Model followed by the rise of lean and agile thinking. One of the most popular agile methods known as Scrum is presented in further detail in order to provide perspective to Kanban. Understanding the rigorous and absolute nature of Scrum highlights how Kanban deviates from other agile methods by being "softer" in nature.

One can observe a major shift in the attitudes and development methods within the last decade: the shift from predefined development scope, budget and schedule to permissive development, which endorses incremental and iterative updating of customer preferences. Supporting this view, McCauley (2001) argues that the fundamental philosophy of a process driven software development is not feasible with its frozen backlog requirements. Thus, there is a need for a more feasible and definitely adaptable and method.

The history of software development is crucial in the process of understanding how different set of values and techniques Kanban presents. In order to comprehend the potential pitfalls of Kanban, the previous and co-existing conducts are essential to highlight the limitations and deficiencies that Kanban might hold within.

2.1 THE WATERFALL MODEL

The oldest software development model goes all the way back to 1956. Back then Herbert Benington held a presentation describing phases in software development process, which was later given the name Waterfall Model. The Waterfall Model describes the nature of a software development process flowing increasingly downwards through pre-stated phases of a development chain (Larman & Basili, 2003).

Benington's (1956) model to create working software includes nine stages. The stages consisted of the following: operational plan, machine/operational specifications, program- and coding specifications, coding itself, parameter- and assembly testing, shakedown and lastly system evaluation. As Figure 1 illustrates, all the phases had to be conducted one after another and only after the previous phase was finished in order to create working high quality software (Larman & Basili 2003).

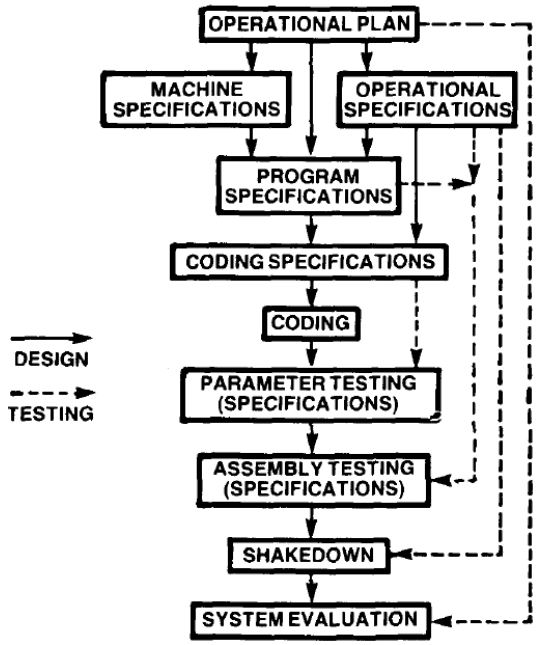

FIGURE 1 Program production visualization by Herbert Benington (1956).

In 1970 Winston Royce described similar phases as a part of software engineering process respecting the constraints of government contracting. The constraints required clear statements to the scope of the project, budget and schedule. Today, these constraints are referred to as iron triangle trap.

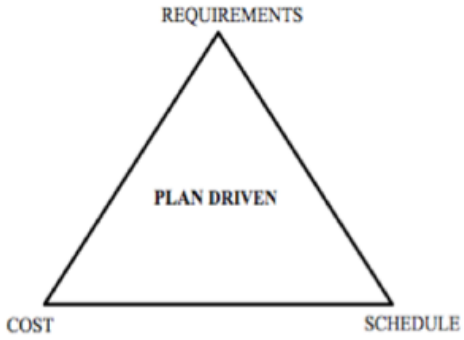

FIGURE 2 Iron Triangle Trap (Leffingwell, 2007)

The phases of Royce's process model resembled the shape of a waterfall, flowing downwards, addressing the following phases: system requirements, software requirements, analysis, program design, coding, testing and operations. See figure 3. Only when the preceding phase was completely done could one move on to the next phase. The development phases were cascaded to each other in a way that allowed the next phase to begin only when a predefined set of goals were achieved for the first phase. All requirements were determined up front with

the assumption that they were known up-front and remained unchanging (Royce, 1970; Bell & Thayer, 1976; Bassil, 2012).

Even back then in the 1970s the model was perceived flawed and risky inviting failure. According to Royce (1970) himself, the model did not allow reflection or revision once a task had entered the testing stage. The paradigm behind the model originated in construction- and manufacturing industries, where slightest changes after completion were expensive. Even so, the model gives an illusion of controllable and measurable process, with simple document driven milestones.

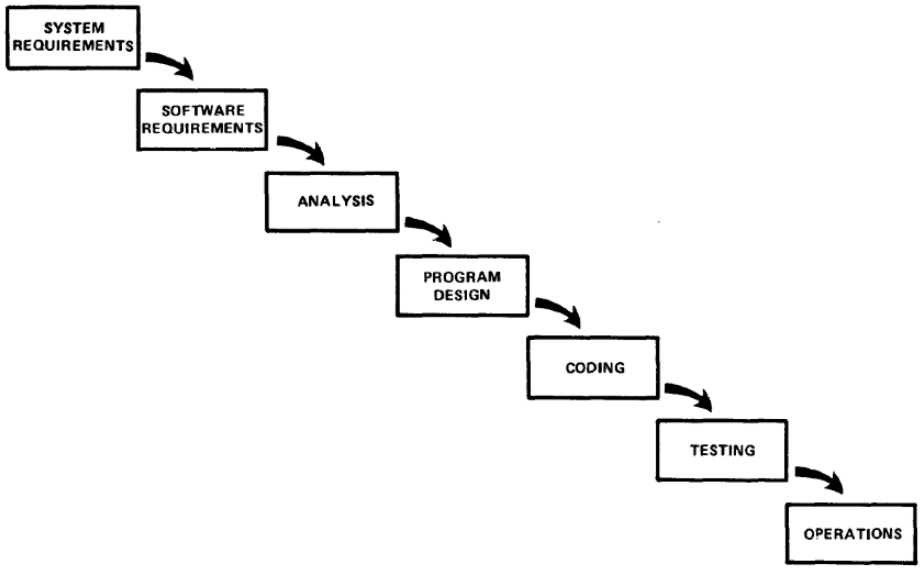

FIGURE 3 An illustrative example of the Waterfall Model (Royce, 1970)

The Waterfall was largely adopted throughout the software industry even though Royce introduced an alternative version, which was considered to be more adaptive and realistic. The alternative version was amended with five additional features decreasing risk. Despite the decreased risk factor, the non-iterative version survived and became the standard model to be used in all big scale software projects across the industry (Leffingwell, 2007). The traditional Waterfall Model looked more appealing to managers since it included early identification of all the requirements. Identifying and locking down the requirements in an early stage facilitate the definition of scope of the project. This in conjunction with managers' need to plan and budget the project were the decisive factors leading to adoption of a model known to be unrealistic and risky. Furthermore, the Waterfall supports heavy documentation, which support data for managers future decision-making. For example, all documentation has to be finished before the actual work can be started to ensure that architecture supports coding.

Succeeding, each line of code has to be ready before testing can take place (Kee, 2006; Jonasson, 2008).

Waterfall has been criticized for its inability to react to changes. Waterfall does not allow any changes to the predefined scope of the project. Everything is done from start to finish without any testing before the testing stage. Defects become visible only when the project is nearly finished making it impossible to react to the mistakes. The possibility that the final product does not match the customer's need after such a long feedback loop increases.

2.2 LEAN SOFTWARE DEVELOPMENT

"The goal of lean is to deliver the maximum amount of value to the customer in the shortest possible time frame." - Dean Leffingwell, 2011

Lean thinking was born as part of the industrial renaissance in Japanese manufacturing after the Second World War. Lean software development is said to be a translation of lean manufacturing into the software context (Takeuchi & Nonaka, 1986; Womack et al., 1990; Liker, 2004). Defining lean is challenging since the concept of lean has been widely used and there are several interpretations of it (Ohno, 1988; Morgan & Liker, 2006; Womack & Jones, 1996, Liker, 2004). Ultimately the purpose of lean philosophy is to maximize value and minimize waste. To simplify the core idea, lean is about 'doing more with less' by ideally producing 'the right things, at the right time and in the right place'. Concepts, such as customer centricity, value stream, waste reduction, workflow analysis and continuous improvement are the main characteristics lean (Womack et al. 1990). The proceedings of the first Lean & Kanban Conference for software development show that lean thinking is becoming largely adopted and spreading around the industry (Willeke et al., 2009; Shalloway et al., 2010).

In a manufacturing context, lean has been able to reduce error rates to one per million units, which makes lean very attractive to software practitioners. Furthermore, lean had been shown to have the potential to at least double productivity, to significantly reduce lead-time and reduce the overall cost (Womack & Jones, 1997; Shingo & Dillon, 1989; Middleton & Joyce, 2012). In a BBC Worldwide Case study, Middleton and Joyce (2012) presented improved company figures after integrating lean methods into software processes. The results indicated

that over the last year, systems lead-times had improved on average by 37% and the consistency of delivery rose by 47%. Moreover, the amount of defects fell by 24%.

As stated previously, the term lean has several interpretations and therefore has no formal practices. Due to the limited scope of this study I will present three lean interpretations that support each other by respected lean literature authors. To begin with, here is a list of guidelines provided by Mary and Tom Poppendieck (2003) that support the values and principles of lean thinking.

1. Eliminate waste, understanding first what value is.
2. Build quality in, by testing as soon as possible, automation and refactoring.
3. Create knowledge, through rapid feedback and continuous improvement.
4. Defer commitment, by maintaining options and making irreversible decisions in the last responsible moment when most information is available.
5. Deliver fast, through small batches and limiting WIP.
6. Respect people, the people doing the work.
7. Optimize the whole, by implementing Lean across an entire value stream.

The most recognized principle of lean thinking is the elimination of waste. The goal is to eliminate non-value adding tasks, the so-called waste. Eliminating waste can refer to useless meetings, tasks and documentation, partially done work, extra features, relearning, handoffs, task switching, delays and defects (Poppendieck & Poppendieck, 2003; Ohno, 1988). One of the most interesting forms of waste in lean thinking is multitasking. Moreover, only tasks that create value immediately and not in the future should be undertaken (Kniperg, 2014). The reasoning behind the concept is that business circumstances change constantly. The project might end up not needing the “task” after all. Work that might create value somewhere in the future should be pushed aside and time should be spent only on activities that explicitly add the value of the product (Poppendieck & Poppendieck, 2003).

Another core activity of lean addresses the limitation of work. With integrated work in progress (WIP) limits, lean software development pulls work into the system only if capacity is available. Womack and Jones (1997) describe the pull effect as the opposite of the

traditional “push method” in which case work is basically pushed into the system regardless of the capacity available. Hopp and Spearman (2001) elaborate that by limiting the amount of work in the system deliberately lean exposes flaws of a process. The mathematical basis for lowering WIP in order to reduce lead-time has been described in the following chapter.

In addition to waste elimination and WIP limitation, Liker (2004) describes continuous improvement and the visualization of work as the most integral and salient characteristics of lean. Visualization is an indicator of how the process is performing. Moreover, visualization helps to manage the process as a whole. Lean studies the process efficiency from a bird perspective optimizing the whole instead of singular activities. To perceive the system as a whole, organization has to have a deep and extensive comprehension of lean thinking and principles related to it (Poppendieck & Poppendieck, 2003).

The second perspective to lean is provided by Womack and Jones. Womack and Jones (1996) list down five core properties of lean thinking that are to induce a lean outcome in the organization.

1. Value: To understand value from the perspective of the customer and being able to define it is in the heart of lean thinking. One should concentrate on maximizing the value and minimizing the amount of waste. The goal is to make organization deliver as much customer value as possible.

2. Value Stream: An optimized end-to-end collection of actions that are needed in order to deliver the software from the backlog to the customer.

3. Flow: Eliminating discontinuities in the value stream and enabling smooth delivery is in the heart of lean. Womack and Jones emphasized the meaning of flow optimization as one of leans core properties.

4. Pull: Lean is built on a pull system, which undermines the needs of the customer and market as the primary decision-drivers.

5. Perfection: Perfections implies to continuous improvement that should be a enterprise-level concept. At the same time, eliminate all wasteful activities and minimize the amount of defects.

The third point of view presented in this study aims to clarify the main objectives of lean and means to do it. The House of Lean by Leffingwell (2007) has five building blocks:

- The Foundation: Management Support
- Pillar 1: Respect for people
- Pillar 2: Continuous improvement
- Product Development Flow
- And the goal, which is value.

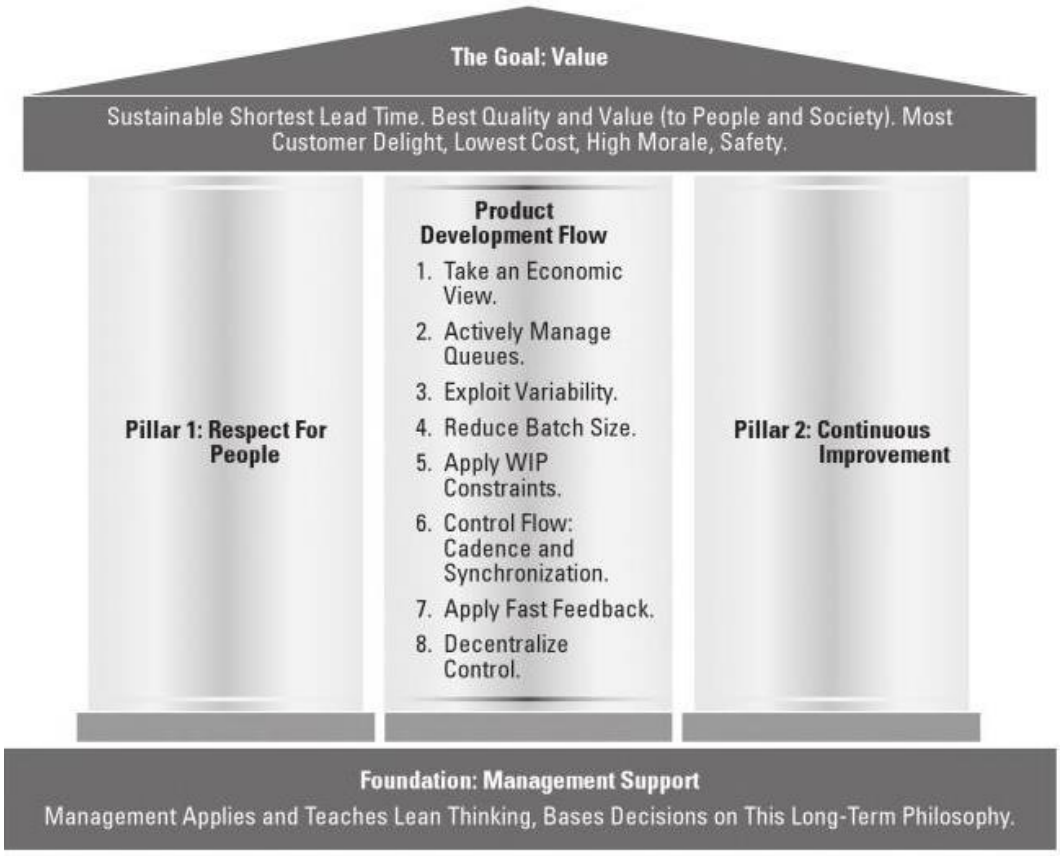

FIGURE 4 The House of Lean (Leffingwell, 2007), adapted from Liker (2004), Larman and Vodde (2009) and Reinertsen (2009)

The foundation of the house and a prerequisite for adapting lean in the first place is management support. Larman and Vodde (2009) argue that a better term to describe the nature of management support is "leadership", where the top management is engaged to lean values and promote them throughout the company.

The first pillar is "respect for people". Lean values people conducting the valuable work instead of the requirements. Lean empowers people to decide for themselves, which increases the efficiency of the work (Leffingwell, 2007).

Continuous improvement is the second pillar of the house. In their book, Leffingwell & Reinertsen (2011) refer to the continuous improvement with the term "kaizen", which means "good change". Continuous improvement incorporates problem solving and decision making by going to the source of the problem. The process data should be observed and based on induced results; effective countermeasures should be applied in order to eliminate inefficiencies. Lean also supports defining milestones, where one can reflect on the project.

In the middle of the house, there is a detailed description by Reinertsen (2009), which provides eight fundamental concepts how to approach the product development flow including cadence and synchronization of the flow. The roof describes lean induced benefits such as shorter lead times, lower cost and customer satisfaction among others.

2.3 AGILE SOFTWARE DEVELOPMENT

"An iterative and incremental approach to software development which is performed in a highly collaborative manner by self-organizing teams within an effective governance framework with "just enough" ceremony that produces high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders."- Lapham, 2012

Agile is commonly seen as a counter reaction to documentation driven, heavyweight software development such as the iron triangle trap (Cockburn, 2002). In fact, Larman and Basili (2003) wrote in their article "Iterative and Incremental Development: A Brief History" that agile methods are considered to be the modern replacement of the Waterfall Model. Agile software development encourages rapid and flexible response to change. It anticipates the need for flexibility in the system and backlog (Anderson, 2010). Customer preferences for the most important features in the software are constantly changing along with the evolving business environment. Agile rose to match these needs with faster and lithesome software development processes (Abrahamsson et al, 2002).

Cockburn (2002) defined agile as the use of light but at the same time sufficient set of rules of project behavior emphasizing the importance of communication and human oriented rules. Agile process management is light while still remaining maneuverable and sufficient. The aim is to code small functional batches of the final product in small proportions constantly releasing the features, as opposed to delivering one large software glob at the end of the project. Agile strives for simplicity by keeping the code short. The code is tested continuously

as well as released on steady cadence. A combined list from Anderson (2010) and Ambler promotes the following concepts:

- Adaptive planning
- Evolutionary development
- Early delivery
- Continuous improvement
- Less documentation is possible if not recommendable
- Communication is a critical issue in software development
- Modeling tools are not as useful as usually thought
- Big up-front design is not required
- People matter

Similarly, Cockburn (2002) and Highsmith (2001) agree that the value from agile stems from recognizing people as the primary drivers for successful projects together with an intense focus on effectiveness and maneuverability. Further, Highsmith et al. (2005) claim that creativity and innovation can only be established if individuals are recognized as the primary source of value. These concepts together yield a combination of values that define what we today consider as the agile worldview.

An opus called the "Agile Manifesto" encapsulates all these values. The manifesto was created by 17 representatives from all different iterative software development disciplines as they came together united by the same goal: to promote agile as the primary approach to software development. The Manifesto stands for the groups' values (Beck, 2001) and it is today considered as the definition of agile software development.

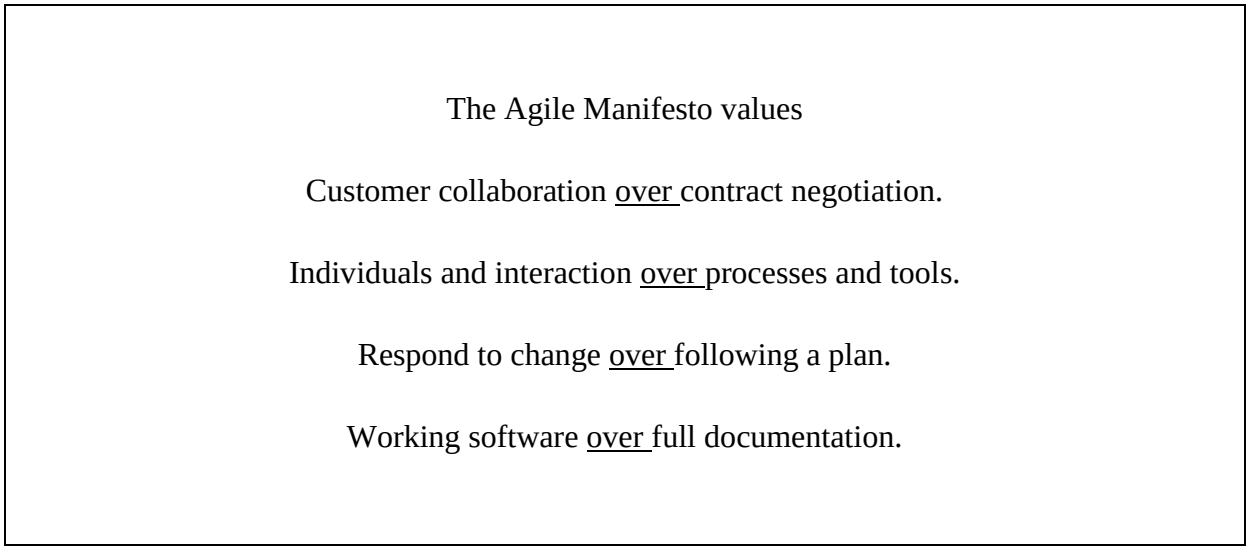

FIGURE 5 Agile Manifesto (2001)

The Manifesto values customer collaboration over contract negotiation. Cooperation and building a good relationship between the developers and the customers is favored over strict contracts (Ambler, 2002). The contract negotiations should be considered as an opportunity to create a trustworthy and stable basis for collaboration. From the business perspective, Abrahamsson et al. (2002) state that agile seeks to deliver business value to the customer instantaneously after the beginning of the project, thus avoiding the risks of contract non-fulfillment.

The Manifesto encourages individual interactions instead of relying on processes and tools. As described before, the value of agile is said to stem from the people. Relationships should be nurtured and communality valued more than institutionalized processes and development tools. The message is to create a tight working environment lowering the threshold for communication and not to rely on predefined methods (Beck et al., 2001; Abrahamsson et al., 2002).

The third point emphasizes responding to change instead of following predicted estimates and preliminary plans. The iron triangle has predefined schedule, scope and budget. As for agile projects, all these aspects have to be loose in order to make software in iterative cycles and be, by definition, agile. Agile development teams focus on features that have the highest priority according to the constantly changing content of the backlog. In agile development even released features can be modified. The rule is that changes in the environment affect the development process (Honkonen, 2014).

The fourth argument states that working software should be the ultimate goal. For Waterfall Model, heavy documentation was often in the center of the attention. Instead of concentrating on reporting, the purpose of documentation, the focus should be on the goal of the whole project: getting high quality working software released. This is the ultimate message of the fourth suggestion. Again, Agile Manifesto does not neglect useful documentation but values working software more (Highsmith, 2001).

In addition, the Agile Manifesto includes 12 agile principles, which naturally reflect agile values and provide concrete advice for agile thinking. See figure 6.

FIGURE 6 Twelve principles of Agile Manifesto (Beck et al., 2001)

The principles of agile software development present the opposite approach to software development compared to the traditional Waterfall. Agile process consists of short iterations where phases such as analysis, design, code and testing iterate in short feedback loops. See figure 7.

Testing of the software is conducted at numerous stages during the development process concurrently with coding exposing mistakes early in the development chain. Since new releasers of software are launched frequently users can validate the value of the new feature. This way better decision about the future preferences can be made based on actual knowledge rather than assumptions. For this reason agile requires customer participation during the iterations in order to prioritize the backlog (Highsmith, 2001). Moreover, Cockburn (2002) states that agile methods are designed to release software rapidly in order to achieve early winnings, fast feedback and to spot defects sooner.

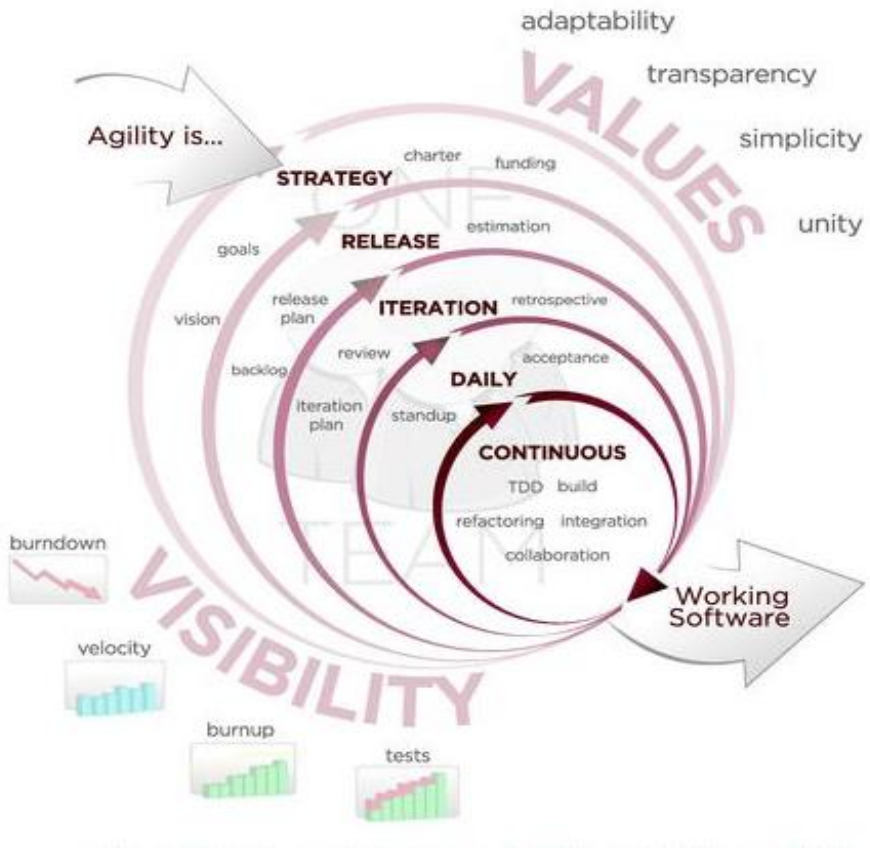

FIGURE 7 Agile Development Model (Sighn, 2013)

The iterative development paradigm can be regarded as a brand of agile process management in which new features can be introduced to the project and old ones removed from the project (Favaro, 2002). This way agile shifts the focus from the process into the software. Supportively, Thomas (2010), Takeuchi and Nonaka (1986) remind that instead of thinking agile as a process, which can simply be adopted, agile should be considered as a target culture.

Note, there are several incarnations of agile but due the scope of this study only Scrum will be presented in order to provide perspective to Kanban.

2.4 SCRUM

Scrum is the simplest idea under the sun: find out what customers really want and continuously deliver that to them sooner. The problem is that implementing this simple idea means unlearning most of what today's managers know for certain, things that are taught in every business school and assumed as fundamental truths in most management textbooks. – Mike Cohn, 2009

Scrum by definition is an agile way to manage a software development project. Schwaber and Beedle (2002) describe Scrum as an approach applying the ideas of industrial process control theory into software development resulting in an approach that reintroduces the ideas of *flexibility, adaptability and productivity*. Scrum is maneuverable to external demands and able to respond to complex and unpredictable changes.

Schwaber (2004) proposes that Scrum should be viewed as a framework guiding the software development. Sharing Schwaber's approach, Gloger (2009) writes that Scrum is much more than a tool. He goes as far as to describe Scrum as a mindset. He argues against the fact that some managers perceive Scrum as a process management tool to control a development process. Although Scrum is described as a framework without specific software mechanisms, it does have a rigorous set of rules to guide the development process. Scrum determines the roles of each team member, scheduled events and other artifacts crucial to the process's success. Schwaber and Sutherland (2012) narrate that the ultimate goal of Scrum is to foster transparency, inspection and adaption.

2.4.1 THE HISTORY OF SCRUM

In 1986, Takeuchi and Nonaka published a today world-known article in the Harvard Business Review mentioning first time the term "Scrum". The article compared a new holistic innovation approach to rugby where the whole team tries to go to the distance as a unit. The paper drew on experience of iterative methods adding adaptive and fast self-organizing teams to the equation of a product development process (Schwaber & Beedle, 2002; Denning, 2012). Later in 1991, DeGrace and Stahl were the first to mention Scrum in a software development context. With the support from Agile Manifesto, Scrum has been largely adopted as an alternative to traditional Waterfall.

2.4.2 THE THEORY OF SCRUM

Scrum advices to divide the development process into series of sprints. Most sprints are time boxed to last from one to four weeks. During a sprint, the Scrum team makes a predefined set of features from the idea to functionality. The features are coded, tested and integrated into the evolving software/product. By the same token as sprints, Scrum invites discussion on integration of information by arranging daily scrums and retrospectives that enable fast feedback and identify new opportunities (Schwaber, 2004).

The Scrum product backlog is similar to the one of Kanban, as we will see later on. It is an ordered list of everything that the customer requires for the product. In Scrum the product owner is a person who is responsible for managing the product backlog and prioritizing it in a way the team is always working on the most valuable feature. The product owner maximizes the value of the product by interacting with stakeholders about the features to be added to the backlog. Using Scrum methodology, a way to fill the backlog is to populate it with user stories, which are short descriptions of functionality described from the perspective of a user. A subset of the whole backlog is called sprint backlog. It can be seen of as the team's to-do list for the sprint whereas a product backlog is a list that includes a list of features to be built (Schwaber, 2004).

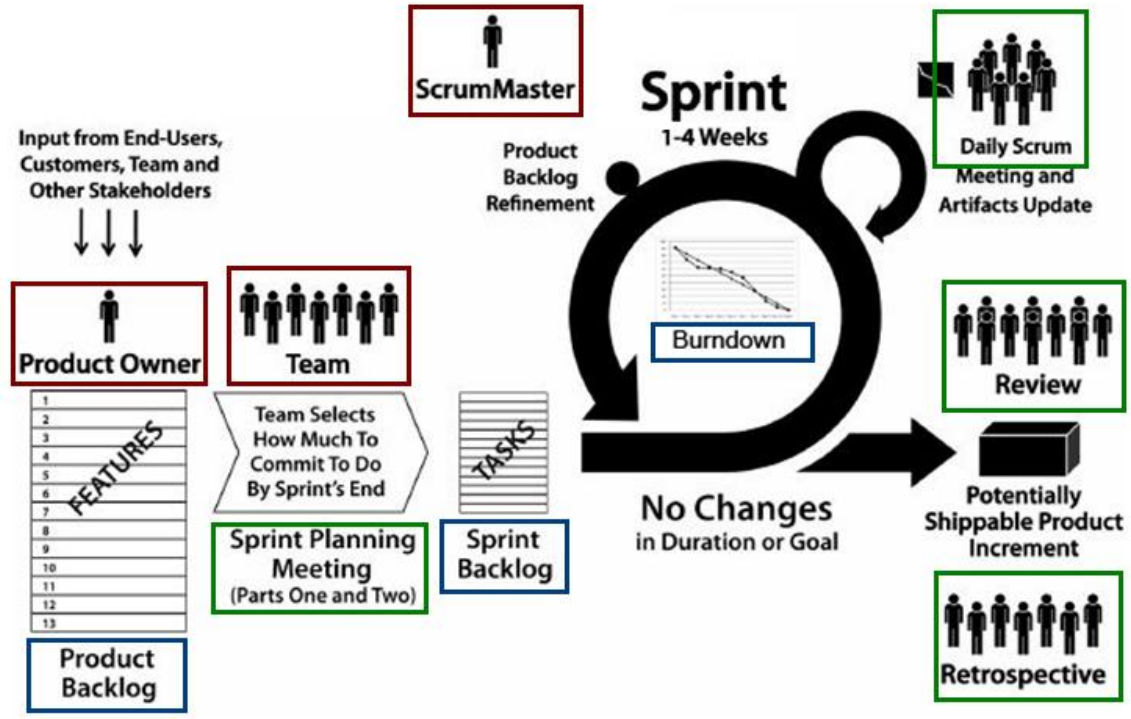

FIGURE 8 Scrum Process (Urbanski, 2013)

Equally important, Scrum introduces a new role of Scrum Master. Scrum Master is responsible for Scrum to be successful by ensuring that Scrum values, practices and rules are followed and embraced. Scrum Master serves the development team by coaching them, facilitating and organizing sprints and other events. Another important task of Scrum Master is to remove impediments ranging from organizational bureaucracy to competing external requests. Scrum Master differs from a traditional project manager in many ways. One of which is that Scrum Master does not provide day-to-day guidance or assign tasks. Instead, a competent Scrum Master protects the development team from outside distractions, allowing team members to focus their efforts on the sprint backlog. While the Scrum Master's focus is on helping the team be the best version of itself, the PO navigates the team towards the goal by conveying that vision through the product and sprint backlog (Schwaber, 2004; Cohn, 2009).

In the light of agility, the roles and the titles inside the development team are void. The team is expected to swarm around tasks. Even though team members join the team with various skills and job descriptions for Scrum those definitions are insignificant. According to Scrum methodology "each person contributes in whatever way they can to complete the work of each sprint". With Scrum the needs of the team are put first and individuals occasionally have to work beyond their preferred disciplines and comfort zones if it is beneficial for the team (Schwaber & Beedle, 2002; Schwaber & Sutherland, 2013).

The additional artifacts of Scrum include the sprint burn down chart and release burn down chart. Burns down charts reveal the amount of work remaining. They are powerful tools, which help to determine whether a sprint is on schedule and can finish all the planned work by the due date (Cohn, 2009).

Embracing the rules, Denning (2011) presented a list to help with the implementation.

1. Organize work in short cycles
2. The management does not interrupt the team during a work cycle
3. The team reports to the client, not the manager
4. The team estimates how much time work will take
5. The team decides how much work it can do in an iteration
6. The team decides how to do the work in the iteration
7. The team measures its own performance

8. Define work goals before each cycle starts
9. Define work goals through user stories
10. Systematically remove impediments

To conclude, Schwaber and Sutherland (2012) make an illustrative comparison of Scrum and a racecar. Recalled, Scrum team has three roles: product owner, Scrum Master and the development team itself. The scrum team is the car itself, ready to speed along in whatever direction it is pointed. The PO is the driver, making sure that the car is always going in the right direction. Finally, the Scrum Master is the chief mechanic, keeping sure that the car is well tuned and performing at its best.

2.4.3 SCRUM INDUCED BENEFITS

Scrum aims to improve the existing engineering practices by implementing steady cadence feedback loops and consistently identifying and removing impediments in the development chain. At its best the process should be transparent and adaptive with frequent inspection points (Cohn, 2009). According to Schwaber (2004) Scrum can ultimately increase productivity rate, induce higher quality and simultaneously reduce lead-times. At its best, Scrum can leverage the innate traits and characteristics of people allowing them to do great things together (Mayer, 2009).

However, so far studies of agile methods have concentrated on project outcomes, failure vs. success, rather than on individual benefits agile promotes in an organization (Scrum Alliance, 2013). Many respected authors have written posts about the outcomes of Scrum but surprisingly little has actually been empirically studied. Yet, Denning (2011) exemplifies the power of Scrum with a company called the Salesforce.com, whose results grew by 41% over a sustained period of time after implementing Scrum. The CEO of Salesforce.com, Marc Benioff, was later chosen as the most valuable CEO on the planet by Forbes (Denning, 2011).

3. KANBAN AS A SOFTWARE DEVELOPMENT METHOD

A number of kanban[3] *equivalent to the agreed capacity of a system are placed in circulation. One card attached to one piece of work. Each card acts as a signal mechanism. A new piece of work can only be started only when a card is available. This free card is attached to a piece of work and follows it as it flows through the system. When there are no more free cards, no additional work can be started. Any new work must wait in a queue until a card becomes available. When some work is completed, its card is detached and recycled. With a card now free, a new piece of work in the queuing can be started. – David Anderson, 2010*

The third chapter introduces Kanban; what it is, how it works and what it does, the values and core properties it stands for, and the goals that it strives to near. This chapter will also present the short history of Kanban and essential theories behind the method. The purpose of the chapter is to provide a basic understanding of what Kanban is, a change management framework, and explain how it enhances process performance and creates the much desired "kaizen" culture.

3.1 THE HISTORY OF KANBAN

Taiichi Ohno developed Kanban after the Second World War in Japan for manufacturing purposes while working at Toyota. Kanban's goal was to promote improvement and maintain a high level of production (Ohno, 1988; Gross & McInnis, 2003). David Anderson appraised the shift towards agile and lean thinking in software development. He was the first one to apply Kanban into software development context in the year 2006 and is today considered as the creator of Kanban in the software development context.

At the beginning of the millennium Anderson had two main challenges concerning the work of a software development team. First, how he could protect the team from incessant business-demands and sustain a steady cadence in software deliveries. The second challenge was more comprehensive: how to scale adoption of an agile approach to cover the whole organization and overcome resistance to change. In order to respond these challenges,

[3] Originally the word "*kan-ban"* written with small "k" refers to a Japanese word that literally means "signal card".

Anderson created Kanban. In his book from 2010 he explains how he developed the method by synthesizing lean and agile techniques including visualization, continuous improvement, just-in-time production and limiting the work in progress. The theory of Kanban is a patchwork of existing theories that Anderson felt were the most effective combination of improvement tools or to manage process flow. After taking a patch of each supportive theory and tailoring them with his own expertise, Anderson integrated the patchwork into the Japanese Kanban System used for manufacturing (Anderson, 2010).

Before going into depth with the theory of Kanban in software development, one has to understand the differences between different forms of spelling. The spelling determines the concept and scale of Kanban. *"Kanban Method",* which is usually shortened to *"Kanban*" with capital *"K"* refers to the evolutionary change framework that in turn utilizes the *Kanban System,* a board trough which cards flow through. The Kanban System utilizes a pull system, visualization and other tools to catalyze lean outcomes within the organization (Anderson, 2010). In this study I speak of Kanban and by this term, I am referring to the tools, values and methods of Kanban in a software development context.

3.2 THEORIES BEHIND KANBAN

The theory of Kanban is easier to understand and appreciate by first understanding the theories it is based upon. The following section "Kanban Method" will elaborate the values and principles of the method. Here is a list of the dominant theories that set the structure for the theory of Kanban:

- Theory of Constraints & Drum-Buffer Rope
- Just-In-Time production
- Six Sigma
- Statistical Process Control
- Little's Law
- The System of Profound Knowledge

By taking only parts of these theories and combining them in a unique manner, Anderson's goal was to make an incremental improvement system, which would raise the efficiency of the whole development system.

3.2.1 THEORY OF CONSTRAINTS

The theory of Constraints (TOC) was developed by Eli Goldratt in 1984. TOC is a management paradigm that views a system through its “weakest link”, the constraint, which dictates the ultimate performance level of a restrained process. Constraints can refer to people, missing supplies, lack of information, defect equipment or even management policies that suppress the process. These are restrictions that prevent the process from maximizing its performance and finally, reaching its goals (Goldratt, 2004).

Goldratt (1990, 2004) claims that each system, even the most effective one, has a constraint, which limits its performance. More precisely, a system can have only one constraint at a time. Other weak areas are "non-constraints" until they become the weakest link. The TOC uses buffers to “protect” the constraint by placing them just before the governing constraint. This way, the buffer works as a shield against variation in the rest of the process. Thus, ensuring that the constraint is never starved. In some cases, buffers can also placed behind the constraint to prevent downstream failure from blocking the constraint's output (Schragenheim & Dettmer, 2000).

The theory has since been added framework called “Five Focusing Steps” to help identify the constraint and removing it

1. Identify the constraint.
2. Decide how to exploit the constraint.
3. Subordinate everything else in the system the decision made in step 2.
4. Elevate the constraint.
5. Avoid Inertia. Go back to step 1.

Anderson has taken TOC’s five steps, which have proven to fit well to flow problems which are common in software development. The list is based on Goldratt’s version from the year 1984 but the spelling of the steps has been slightly modified by Anderson (2010).

3.2.2. DRUM-BUFFER-ROPE

Drum Buffer Rope (DRP) is an application of the Theory of Constraints known as the pull system. DBR is the primary foundation, which Anderson (2010) used in his previous work before putting together what we today refer to as Kanban. He saw DBR as a possible solution

to his goal to incrementally improve a process by identifying a constraint after another and repeating this again and again. He also made a remark that only workers at the constraint workstation would remain fully loaded with work and other stations would experience slack at the time. The idea of limiting work and exposing constraints, referred to as bottlenecks in the Kanban literature, have created the basis for Kanban's pull system. Anderson has however made some big scale alternations in order the process to recover more gracefully from a constraint (Anderson, 2010).

The DBR application describes how buffers should be used to protect the governing constraint. DBR is the means of TOC to protect the weakest link in the system. Buffering the constraint protects the whole system against variation by decreasing process dependency. Thus maximizing the efficiency of the process by decreasing inventory and defects, which enables making more high quality products. Ultimately, the aim is to create a reliable due-date performance, make effective exploitation of the constraint and make response time as short as possible respecting the constraint (Schragenheim & Dettmer, 2000).

3.2.3 JUST-IN-TIME PRODUCTION

Lean thinking is said to have emerged from the Toyota Production System (Womack et al., 1990) and since Kanban is a lean software development framework, many of Kanban's core values have roots in TPS. While the previous chapter already explained the core values of lean, this part will show how just-in-time (JIT) production, which is an essential part of TPS, is adapted by Kanban.

JIT signifies producing only what is needed, when it is needed and the exact amount how much is needed. The goal is to increase a company's return on investment by decreasing the existing inventory levels and cost associated with it. Along with the decreased inventory levels, the amount of waste and variance should decline and result in improved quality and efficiency. Furthermore, JIT process utilizes signals between different work phases in order to communicate. Signal cards tell the upstream process to start working on the next part (Schonberger, 1982; Toyotaglobal.com).

Anderson has taken the fundamental idea behind JIT and made it to fit his own criteria improving the flow of knowledge work. In software development context, JIT means deciding as late as possible in order to maximize existing knowledge. Kanban also limits amount of

work in the process and uses cards to communicate. The next chapter discloses how the signals have been further applied to Kanban.

3.2.4 SIX SIGMA

Six Sigma is a collection of tools and techniques for process improvement developed by Motorola in 1986 (Geoff, 2001). Today, Six Sigma is used across industries. More than 50% of the Fortune 500 companies are utilizing Six Sigma in their organizations (Marx, 2007).

Six Sigma seeks to improve processes by identifying and removing causes of defects. The ultimate goal is to create a stable and predictable process by decreasing variability, both special- and common cause. Six Sigma relies on measurable process characteristics, which can be analyzed, controlled and hereby improved. Due to the measurable nature of the improvement process, the produced data can prove quantifiable financial returns that Six Sigma induces (Dedhia, 2005).

The vantage point of Six Sigma lies in its framework. It facilitates the application of tools and techniques in a structured and disciplined manner guided by data driven decision-making (Lee-Mortimer, 2006; Inozu et al., 2006; Hahn et al., 1999). Six Sigma aligns projects to their strategic objectives. The projects are measured with both statistical and non-statistical quality improvements tools and integrated into a problem-solving framework. The framework has clearly defined performance measures, which indicate the level of the process performance (Dedhia, 2005). Six Sigma projects aim to reduce the defect to a maximum of 3,44 errors per million exposures (Harrington, 2005). Equally important, Hahn et al. (1999) point out that the process improvement requires commitment from the entire organization in order to sustain the achieved quality level. Six Sigma emphasizes the importance of top-level management support and management's role as decision makers based on the produced verifiable data and statistical methods rather than assumptions.

3.2.5 Statistical Process Control

Statistical Process Control (SPC) can be traced all the way back to 1920 to Walter Shewhart. Another name along with Shewhart combined with SPC is Edward Deming who is considered as one of the fathers of quality management (Barlow & Irony, 1992; Bergman, 2008).

Compared to Six Sigma, SPC is more of an explicit method rather than a management approach. Six Sigma can be regarded as a higher-level concept whereas SPC is a tool that Six Sigma utilizes.

SPC monitors the flow of a process while trying to decrease variance and improve the overall efficiency and predictability. SPC activities can be divided into three phases. In the first phase, the process and the specification limits have to be understood. Control charts measure variation visualizing a process map with points indicating variation. Variation can be divided into two categories, common and special variation sources. The common sources are predictable and a part of the process. Special sources are unexpected events that can result in special unpredictable variation. A process is said to be unstable and out of control if it exhibits special cause variation. In addition to being harder to detect, special cause variation is more expensive (Howard, 2003). Therefore, the second phase concentrates on eliminating special sources of variation in order to stabilize the process. A process is stable if the control charts do not have any unpredictable peaks, which would trigger a so-called control chart detection rule. However, if a process seems to have unexpected peaks or lows and the control chart considers this as excessive variation, SPC uses tools such as Ishikawa diagrams, designed experiments, and Pareto charts to identify the source. In the third phase SPC monitors the process with control charts to detect any significant changes and sources of variation (Oakland & Followell, 1990).

3.2.6 Little's Law

Little's Law is a theory for queuing proven by John Little in 1961. According to the theory, the average number of customers in a stable queuing system over any period of time is equal to their average arrival rate, multiplied by their average time in the system. This rather simple result is quite remarkable since it follows that the behavior is entirely independent of any of the detailed probability distributions involved. Hence, behavior requires no schedule

assumptions about the customers who arrive or are serviced in the system. Excluding also the facts whether they are served in the precise order in which they arrive (Little's Law, 1961).

For this study Little's Law is essential because it supports Kanban's core value of reducing the amount of work in the process. Little's Law proves that the less work is in the process, the faster the work is ready. Little's Law can be converted into an estimate of an average lead-time. Here is an elaborative example.

$L = \lambda W$
L is the number of works/tasks in the process on average
λ is the effective arrival rate of the work on average
W is the average time that the task spends inside the process e.g. lead-time.

Therefore, the average lead-time is $W = L / \lambda$. If the number of tasks (L) in the system is 2 and every hour 10 new tasks arrive (λ) we can calculate that $W= 2/10 = 0,2$ which means that the average lead-time is 0,2 hours. If there are 4 tasks in the system, 10 new tasks arriving every hour, the average lead-time is $W = 4/10 = 0,4$ which means that the average lead-time is 0,4 hours. The relationship is linear. The numbers show that the less the team is working on the faster they can finish it. Significantly longer lead-times seem to be associated with significantly poorer quality. One should also consider the overhead that shifting knowledge work generates. Anderson (2010) writes in his book that approximately 6,5 times increase in average lead-time resulted in a greater than 30 fold increase in initial defects. Although the relationship between average lead-time and WIP is linear, the relationship between WIP and quality is not. That is, defects disproportionately increase in relation to the quantity of WIP. With this evidence, managers can argue the reduction of WIP in order to improve quality. With shorter iterations release times conveyed from lead-times will contribute to higher quality. Moreover, shorter iterations and releasing working code every two days instead of four builds trust with external partners (Anderson, 2010).

3.2.7 The System of Profound Knowledge

The System of Profound Knowledge (SoPK) is considered to be a culmination of Edward Deming's lifelong work published in his final book in 1993. SoPK beholds an organization as a system. It defines a system as a network of interdependent components that work together

trying to accomplish the aim of the system. According to Deming, the aim for any system should be that everybody wins not excluding any part to pay the expenses of others. From the business point of view, the system includes all the stakeholders in the value chain. Deming (2000) used the analogy of an orchestra to illustrate the concept of a system: "An orchestra is judged by listeners, not so much by illustrious players, but by the way they work together. The conductor, as manager, begets cooperation between the players, as a system, every player to support the others. There are other aims for an orchestra, such as joy in work for the players and the conductor." When the system is optimized as a whole SoPK claims that significant results can be achieved.

Deming (2000) accounted for four parts that all managers should have. First, all managers should have appreciation towards the system understanding the overall processes involving all the stakeholders. Second, managers should recognize the difference between special and common cause variation more commonly referred to as the Knowledge of Variation. Each system consists of both normal variation and of special cause variation as described in the previous section. Similarly, Deming teaches that normal variation should be spared and managers should concentrate on eliminating special cause variation in the process. The third part managers should understand is the basic Theory of Knowledge. Theory of Knowledge explains concepts and limits of what can be known. The last part Deming listed, as managers' core know-how was the understanding of basic psychology: how people behave and think. Deming also noted that these four parts are inseparable.

As the chapter of Kanban will further illustrate, one the main assumptions that Kanban sets of with is that the process is seen as a whole. Instead of considering each phase as an individual unit in the process Kanban tries to smoothen the flow through the entire system.

3.2.8 Theory Synopsis

All of these theories adapted to Kanban build a unique method to incrementally improve software process efficiency. The Theory of Constraints together with Drum-Buffer-Rope shift the focus to bottlenecks that prevent the process from performing to its highest level. TPS together with lean thinking brings along the concept of waste elimination, turning the focus on doing only the right things. Further, Six Sigma and SPC give explicit tools to manage variance and means to increase process predictability and flow. Just-in-time thinking shifts

the decision-making to the latest point possible maximizing the amount of existing knowledge. In addition, Little's Law creates the foundation for better quality by proving that reducing WIP reduces lead-times, which decreases the number of defects. Anderson integrated these theories with the Japanese manufacturing flow framework creating what we today refer to as the Kanban Method in software development.

3.3 KANBAN METHOD

The purpose of this chapter is to provide a sufficient understanding of Kanban summarizing the core principles, properties and tools, which in turn enable the reader to understand the potential pitfalls of Kanban. The purpose is not to teach how to implement Kanban or present Kanban in-depth, but to come up with a sufficient synopsis of Anderson's book "Kanban: Successful Evolutionary Change for Your Technology Business" published in 2010.

Instead of giving a straight definition of the Kanban Method, Anderson is hesitant since he feels that the theory and method are not permanent but gradually evolving. Yet, he argues that Kanban Method is not a software development lifecycle methodology nor is it an approach to project management. By this, he refers to the fact that Kanban alone is not sufficient as a process management tool. Kanban requires an existing process so that it can be applied incrementally to improve the underlying process. Thus, it is a tool for continuous learning and optimizing workflow within a process. As phrased by Anderson, Kanban is an approach to incremental, evolutionary process change, which can be overlaid only on an existing process exposing bottlenecks and introducing positive change over time.

Kanban is a pull system that pulls work into the system only when the system has the needed capacity to handle the work. The system is visualized with a board where work flows through the system from left to right undergoing the phases needed to finish the task. All Kanban boards, systems, are different. Kanban actively encourages diversity and tailoring Kanban to fit the situation and nature of customer demand. Anderson said that what Kanban does is to give the market the permission to ignore best practices based appraisal schemes creating a system that is only guaranteed to work for that specific situation. Against the common practice in software industry, Kanban supports tailored approach rather than dogmatically following a lifecycle process predefined by a template.

Kanban is way to prioritize tasks and clear the focus for the development team. Tasks are on chosen from the product backlog, which consists of features, bug fixes, non-functional requirements, to enter the pull system. The meeting to prioritize the backlog should involve all the people that are involved with the project. The size of the backlog directly affects the overall lead-time and therefore the prioritizing meetings should be held at a steady cadence.

Anderson illustrates Kanban with three leading principles and five core properties of the theory. The three leading principles answer to question how one should think, whereas the five properties provide action points, how to actually do it.

3.3.1 THREE PRINCIPLES OF KANBAN

1. Start with what you do now

Kanban starts evolving the existing process instead of replacing it with a completely new one by making radical changes. Kanban is not, as Anderson said, a software development tool but a framework for change management.

2. Agree to pursue incremental, evolutionary change

Supporting the first principle, the development team must warrant an evolutionary approach to improvement. Change always creates resistance and only way to safely implement and make adjustments to the process is to guarantee a slow and gentle approach.

3. Respect the current process, roles, responsibilities and job titles

Kanban respects the current process by admitting that the current process potentially has elements worth preserving and embracing. Supporting the previous principles Kanban's third principle seeks to decrease change resistance by maintaining current work roles. The people in the process have their own opinion and a current role in the system, and are therefore hesitant to change anything that might worsen their position in the company or add their workload. Therefore, Kanban is in favor of incrementally taking steps towards the evolutionary change.

3.3.2 FIVE CORE PROPERTIES OF KANBAN

If the principles of Kanban gave advice how to approach a project creating an optimized mind set, then the properties of Kanban are guidelines on how to manage the change.

1. Visualize Workflow

Software development is knowledge work and consequently intangible. Customer requests are abstract, which are made into intangible products. Honkonen (2014) compared the situation to a cobbler repairing shoes. A cobbler can visually see how many shoes he has left on the shelf waiting to be repaired. A software developer does not have anything concrete, only a request for a new feature in the software. The overall work load visibility is very poor given that software is usually done in teams, which further complicates digital work queues. As a result, a team or an individual coder can have a hard time understanding when and where the work is being done. Honkonen argues that there is need for more comprehensive visualization that provides an overview of the situation. And with a better overview, relevant discussion and better decisions can me made while spending less time pondering over irrelevant issues.

Kanban uses a board with cards and columns to visualize the process. The columns on the board represent different phases of the workflow. The card, post-it[4], represents a work item[5] as it flows through the development process represented by the columns, from left to right. See figure 9. A work item can refer to a customer request, which can be e.g. a new feature to the existing software or defect correction. For example, a post-it in the "pending" column means that the work has been selected to be executed, but is still "pending" to proceed. The numbers at the top of each column limits the number of cards; work allowed in each column (Hiranabe, 2007; Anderson, 2010). This will be explained by the second core property of Kanban.

[4] A Kanban Board with post-its is common sight since post-its can be easily moved into the next column while the task proceeds in the system.

[5] This study uses a work item as a synonym for a user story, epic, task as well for work task.

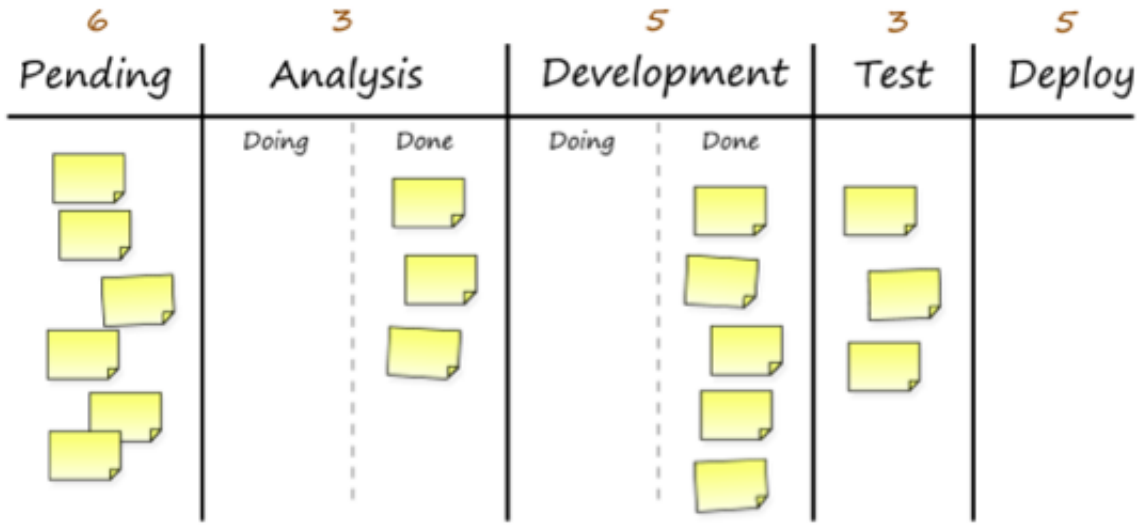

FIGURE 9 Kanban board in its simplest incarnation. A Kanban System consists of a big board on the wall with post-its places in columns with numbers on the top. The numbers signal WIP limits (Hubbart, 2010)

2. Limit Work-In-Progress

Kanban sets limits to the quantity of tasks in the system. More precisely, Kanban restricts the quantity of tasks in different phases of the system. This is called limiting the work in progress, (WIP) which is a salient factor in Kanban's success. Unlike the Theory of Constraints, Kanban methodology uses the term "bottleneck" for expressing a point of constraint: a point where a queue starts to line up, and is slowing the process down. On a highway, a point where construction is done, the traffic slows down to fit only to one lane. Similarly, a phase in the software development chain can slow down the whole system. A constraint, in Kanban methodology, refers to the WIP limit set by the team itself. Limiting WIP creates a pull system for all the phases of the workflow. The work phases have a WIP limitation, which is written above the column. Only when the WIP limit allows new work, a new task, can be "pulled" from the previous phase presuming that the task is done and waiting to proceed. For example, figure 9 tells us that the constraint, WIP limit, for the Analysis phase is three. Altogether only three work items can enter analysis phase, including both "doing" and "done" sections. Doing signaling that the task is still under progress. Done signaling that the task is ready for the next phase of the process to pull it once there is enough capacity

In figure 9, the bottleneck is testing. WIP limit of the test phase is three, allowing only three items to be worked on simultaneously. The development phase allows five items at a time and has finished them all since they are located in the done part. However, the development phase cannot pull more tasks from the analysis phase. WIP limit of five prevents this even though the tasks are done and the workers idle. One can see that the analysis phase is idle as well, since all the tasks locate in the done part. The whole system is slowed down by the testing phase; hence the bottleneck for the whole system has emerged.

One could argue that what good it does that the rest of the workers sit idle doing nothing while testing is scrammed and blocking the stable flow of the items. Why should they not continue to work and increase the WIP limit of the testing phase? At least then the task would be ready for testing. This is where the beauty of Kanban steps in. It can painful for the organization but it is mandatory in order improvement to happen. Kanban's constrains illuminate problem areas in the flow so they can be identified and hopefully resolved. Limiting the WIP highlights the bottleneck and makes it visible to the software team. For this team it is testing. It forces the team to focus on testing since it is the phase needing help. Kanban argues that what good it does to work on tasks if they are never going to be ready because something is clearly wrong with the flow of the process. It also forces the team to concentrate on the tasks they have lined up, focus being on finishing them first before anything new can be started. The mantra of Kanban is "Stop starting and start finishing". This is very important since no customer is actually interested whether the teams has started a task but rather when the task is finished (Honkonen, 2014a). Theory of Constraints and Drum-buffer-rope state that a bottleneck should be elevated by making everything else in the process subordinated. By reallocating a person from the analysis or development phase perhaps. At the moment, both of them are non-constraints and therefore after analyzing the situation this could be an option. Thus, multitalented people are very precious to a Kanban process.

Anderson noted that slack is not a bad thing. It enables innovation and allows workers to reflect on the past, creating an opportunity for process improvement. The pull system is a foundation for continuous, incremental and evolutionary changes to the system. It is critical that the phases are constrained which induces the pull effect when capacity is released. Honkonen (2014a) emphasizes the importance of analyzed and small WIP limits since it forces the team to focus on finishing tasks because constraints prevent the team from starting anything new. At the best-case scenario, the team starts to collaboratively solve the bottleneck and co-operating in order to finish work. The ultimate point is to limit work to finish things

faster proven to be a fact by Little's Law. Little's Law tells us that the lead-time is $W = L / \lambda$. L being the number of tasks in the process and λ being the effective arrival rate of the work. In other words, the number of tasks in the process has a direct influence on the lead-time, which means that limiting WIP directly leads to shortened lead-times. A sub product is the increase in quality.

Limiting the WIP is obviously depended of the process in question but the "best practices" have created somewhat differences of opinion. According to Middleton (2001) the WIP should always be minimized in order to maximize quality. As explained in the previous chapter, the lead-time has been found to correlate with lead-time and since lead-time is directly influenced by work in progress, the limitation of WIP should decrease the number of defects. Ladas (2008) on the opposite perceives the WIP limits more smoothly allowing flexibility based on the process capacity.

3. Measure and Manage Flow

The third core property of Kanban supports incremental change. Endorsing SPC activities, the analysis should be done by comparing the old process performance to the new process performance. Kanban starts with understanding the existing process by measuring and analyzing its performance. How is the value flowing through the system? Where are the bottlenecks that one needs to keep an eye on? After understanding the current state of the process, Kanban proposes conducting minor modification based on the measured data in attempt to improve the performance. The goal is to create a positive change with small steps. In order to take a step, one has to have an overview of the process and of its performance. Afterwards, analyzing the results of the change better or for worse. Repeating this formula over and over again is the source of Kanban's incremental process improvement. Honkonen (2014) recalls something he read "once the first problem is solved, the second one gets a promotion". It is just another way of saying that Kanban is always striving to make progress. In the heart of improvement is the opportunity for regular feedback organized in small loops.

4. Make process policies explicit

The fourth core property of Kanban is to make processes explicit. The property overlaps the previous property by stating the same message from another point of view. In the previous scenario, the emphasis was on how to improve the process. The fourth scenario argues that before the mechanism of a process have to be made explicit, it impossible to hold a discussion

about improving it. The discussion tends to be subjective, emotional and anecdotal before achieving an explicit understanding of the process. However, if the facts are understood and agreed upon, the discussion tends to be more objective and empirical. As stated in the previous section, the overview of the process is crucial before making any changes or attempts to improve it. The existing process has to be fully analyzed and defined. Moreover, the team has to have a mutual understanding of the process and work flow. They have to have a cohesive view how the work proceeds through the systems and what the overall goals are. Only then can the team prioritize tasks in a way they all agree upon. And only then, after understanding why the current systems functions the way it does, can the team make correct adjustments. In cases, where changes are made before making the process explicit, the adjustments can be either unhelpful or even harmful to the system.

5. Improve Collaboratively

As discussed, Kanban encourages incremental and evolutionary changes that create a stable and predictable process. Anderson encourages as the fifth core property that this should be done by using explicit theories, which were described in section "Theories behind Kanban". Kanban utilizes tools from various fields of knowledge to encourage analysis of problems, bottlenecks and weak-points of the system in order to discover solutions. There are various methods but Anderson lists as explicit tools the following theories:

- The Theory of Constraints (exposing bottlenecks)
- The System of Profound Knowledge (variation doctrine foremost)
- Lean approach (acknowledging the concept of waste)

3.4 KANBAN INDUCED BENEFITS

Kanban induces benefits by making incremental modifications into the existing process. Some of the benefits have already been elicited previously in the text. This part summarizes the benefits and highlights dependences between actions that create positive chain reactions.

Little's Law proves that decreasing the amount of work in progress leads to decreased lead-times. The duration of a lead-time further affects the number of defects, bugs in the code meaning that decreased lead-time ultimately improves quality of the product. Kanban cuts down lead-times, which means that code can be released more frequently against demand.

Delivering output at regular cadence builds trust with the customer and along the value stream. Limiting the WIP causes a chain reaction benefiting the whole team and organization. Less time is spent on fixing bugs, which on the other hand leaves more time to concentrate on value adding work. The system is more productive and predictable since need for expedite requests such as crucial bug fixes substantially subside and work is finished. Customer satisfaction is a by-product of enhanced process performance. The team experiences a highly collaborative and functional working environment, where it is empowered to navigate the process to its best ability. The team is able to finish work creating a sense of satisfaction through the feeling of success from a completed task.

To continue, Kanban is highly responsive. Short feedback loops and cycle times enable fast and effective adjustments to the process and to the overall direction. Being able to change customer requests, edit the backlog of tasks, reduces risk. Projects do not have to commit to long-term goals that are inevitably obsolete by the time the project would be ready. The accelerated cycle pace and short iterations illuminate problems and decrease the cost of failure because they are spotted early in the development chain (Anderson, 2010; Liker, 2004).

Agile has done remarkable improvements and achieved benefits far larger than expected (Hathaway, 2009; Anderson, 2010). In his book Anderson presents a case of a company named Corbis, which is a company privately owned by Bill Gates with 1300 employees. Corbis was able to deliver on every promise done by Kanban and more. Anderson uses the company as a case example for benefits induced by Kanban.

3.5 KAIZEN CULTURE

If you are not continually improving, but you are doing all of the other parts of the Kanban method, you are missing the point. It's a little like the concept of "doing" Agile but not being agile. - David Anderson, 2010

Kanban is to gradually shape the process with slight improvements towards better performance. Kanban's core principle "start with what you do know" suggest that the changes should be done step by step hereby trying to keep the resistance to the change at minimum. The challenge is at the same time to maintain a steady delivery cadence and follow the agile

pace with the team. The goal is to reach a state where the system focuses on quality improvement, productivity and increasing customer satisfaction. From a bigger perspective, the pursued goal is to change the organization culture. Hence, the term *kaizen*, which generally refers to continuous improvement, tends to be combined with Kanban. Larman and Vodde (2009) describe the formal practice of kaizen as follows:

1. Choose and practice techniques the team has agreed to try until they are well understood → Master standardized work.
2. Experiment until you find a better way.
3. Repeat forever.

Anderson describes kaizen as the environment, where workers are empowered, free to take action and free of fear. An underlying norm of Kanban is to be tolerant of moderate failures if they are caused by an experiment to improve the process performance. The team can spontaneously swarm around tasks, discuss different options and implement fixes. The tasks on the Kanban board are not assigned to team members, rather team members volunteer. They are discussed together while aiming at the best possible outcome for the whole process. For being able to do this, Kanban requires a highly trusting culture where team members are equal, regardless of their position in organization. Anderson found that usually high-trust cultures have flatter organizational structures than lower trust cultures. Empowerment enables a flatter structure to work more effectively. Hence, Kanban is susceptible to criticism because it breaks the norms of Western business management. Kanban and kaizen decrease the amount of management mid-layers in the organization and many managers are afraid to lose their titles. Although the third leading principle of Kanban is to respect the current roles, over time management titles change are bound to transform.

Kaizen culture values system thinking while making local improvements that enhance the overall performance. "System thinking" means considering the process as a whole and not perceiving the different phases in the development chain such as analysis, development differently. The goal is to optimize the whole process instead of maximizing the efficiency of individual phases. Respecting Deming's lifelong work, Anderson took this approach from System of Profound Knowledge, which works marvelously for optimizing lead-times through the whole system. The SoPK is supported by WIP limitations that expose problems in the

process forcing the whole chain to correct them before producing any more work to be left unfinished.

Anderson has created a list of 8 steps to initiate the cultural change. The steps are designed to transform the team towards a Kaizen step by step.

1. Optimize the existing process.
2. Deliver high quality.
3. Improve lead-time predictability.
4. Improve employee satisfaction.
5. Provide slack to enable improvement.
6. Simplify prioritizing.
7. Provide transparency on the system design and operation.
8. Design a process to enable emergence of a high maturity organization.

4 RESEARCH DESIGN

This chapter will explain how the study is designed. The first section explicates the overall strategy and approach chosen for the study. The second section describes how the data was collected and the motivation behind the chosen methods. Following, the third and fourth sections discuss the reliability, validity and ethics of this study.

4.1 RESEARCH STRATEGY

This research is an empirical study of a rather new phenomenon in the field of software. According to Uusitalo (1991) the subject of an empirical study has to be a real world phenomenon and information has to be required with a systematic method. Supportively, the empirical data collection for this thesis was conducted by a systematic qualitative method due to the scarcity of prior research in the field. Supporting the view of Miles and Huberman (1994) the thesis provides a rich insight to the topic, consisting of explanations and descriptions of the phenomenon.

Hirsjärvi et al. (2009) describe the use of judgment sampling data as one the key characteristics of qualitative research. By using judgment sampling the best suitable respondents can be chosen which facilitate understanding the phenomenon. Judgment sampling the potential pitfalls enables to better understand potential high-risk areas, where Kanban is most likely to stumble into a pitfall. Instead of making statistical generalizations the contribution of qualitative study is the ability to describe a phenomenon in depth instead of making broad assumptions of the whole research field (Koskinen et al., 2005). This approach is favored by this study due to the scarcity of prior research. Furher, Eskola and Suoranta (1996) state that the aim of qualitative research is to describe a phenomenon or an event in order to understand a certain activity or give a theoretically meaningful interpretation to a certain phenomenon.

The acquisition of new empirical information was done by interviewing agile experts. Tuomi and Sarajärvi (2006) recommend using interviews in cases where flexibility is at importance. Supportively, Hirsjärvi et al. (2009) argue that interview is a valuable method because it enables expanding retrieved information. Interview can clarify desired answers while producing illustrative examples. This is especially important while working with an unknown

or less explored theme (Hirsjärvi et al., 2009).

The aim of this study is not to try and wreck Kanban into pieces and obsessively look for inconsistencies in the theory but rather to look for potential pitfalls that experts have faced and shared while applying Kanban. For this reason, I chose to use the theme interview. It supports circling around predetermined themes while allowing modifications and follow up questions to be made. Theme interviews are semi-structured and resemble in depth interviews. They are conducted with certain pre-decided topics and relevant targeting questions while trying to actively pursue the goal of the study (Tuomi & Sarajärvi, 2006). In this case, exposing the potential pitfalls.

Furthermore, the interviews were conducted with an abductive approach. Here the term "abductive" refers to a study, which explores and analyzes the existing theory before conducting the empirical phase of the study. This method creates "fruitful cross fertilization" between the theory and reality (Dubois & Gadde, 2002). For this specific study, this approach was especially interesting since the inconsistencies between theory and reality were also highly potential sources of pitfalls that were parts of Kanban's theory that failed to manifest in reality.

4.2 DATA COLLECTION

The sources of empirical evidence for this study consist of interviews conducted with five industry experts that have comprehensive experience in the field software development. As stated previously, the interviews were led by a theme approach in a semi-structured setting. The interviews were subsequently built on the existing knowledge of Kanban. During the interviews the discussion revolved around premeditated themes while allowing conversations to develop into directions each interviewee saw fit, which generated fruitful insights from different perspectives.

4.2.1 INTERVIEWS

The study data consists of five semi-structured theme interviews with industry experts. Four of the experts worked at Reaktor[6], a leading agile software house in Finland. Reaktor is one of the few companies in Finland that has licensed agile trainers that consult other companies to think and behave agile. The fifth interviewee worked at Fonecta[7], Finland's biggest contact information company. Working for Reaktor I was able to conduct interviews with four of the agile consultants. The interviewees are leading experts in their field, and I am very grateful that they granted me access to their knowledge from decades of work in the software field. The experts have travelled around the world to present in software conferences and have witnessed first hand various agile methodologies discovering where and why they have failed. Although the number of interviewees is limited, the depth of the knowledge plays a key role. Supportively, Gomm et al. (2009) state that the statistical significance does not influence the quality of the study. Dubois and Gadde (2002; 2013) continue that with limited resources it is better to focus energy rather than to grasp the surface. The purpose of this study is to provide an in-depth analysis on a particular phenomenon, Kanban's pitfalls, by having a deep scope. With this intention, the purpose is not statistically list down potential pitfalls but to arouse discussion around topics that arise from the experts' experiences and provide perspective for future research.

The data collection was carried out between 17th and 21th of September 2014. Before the 17th I had contacted the interviewees by phone, further sending them e-mail with the purpose of this study and necessary details of the interview excluding the interview questions. Each interview was conducted separately during the five days. The interview lengths varied between 1 hour and 7 minutes and 1 hour 42 minutes. The average length of an interview was 1 hour and 28 minutes.

---

[6] Reaktor is a creative technology company, which is known for their exceptionally well-functioning services. To achieve the finest quality, they keep design and production tightly together. Reaktor takes pride in speed, accuracy and quality. The company was founded in 2000. In 2014 Reaktor employed 300 people and made 28Meur in turnover.

[7] Fonecta group is a Finnish contact information and media company, which is a part of an international group European Directories. Fonecta's core business consists of offering search engine marketing, digital direct marketing and visibility in different Search Medias. In 2013 Fonecta had 900 employees and a turnover of 177 Meur. Fonecta was commonly considered as the market leader for contact information searches.

The summary of the interviews is presented in the table below. The interviews took place at Reaktor Head Quarters except for one that was carried out at Fonecta's Head Quarters. The interviews were recorded upon the consent of the interviewees and carried out in Finnish.

TABLE 1 Summary of the Interviews

Date	Name	Position	Company	Location	Duration
17.9.2014	Michael Holler	Agile Coach	Reaktor	Helsinki	100 min
20.9.2014	Sami Lilja	Certified Kanban & Scrum Trainer	Reaktor	Helsinki	102 min
20.9.2014	Panu Liira	Principal Consultant	Reaktor	Helsinki	80 min
21.9.2014	Juha Vakkila	Director of Online Development	Fonecta	Helsinki	67min
21.9.2014	Toni Strandell	Senior Software Architect	Reaktor	Helsinki	91 min

The transcription phase took place during the 19th and 24th of September. The transcripts were supported by my personal notes and drawings made by the interviewees during the interviews. The transcribed content was carefully analyzed and manually segregated based on emerging topics of the data. After all the data from all the five informants was segregated, the data could be divided into separate categories and further into subcategories. I used Excel spreadsheets for the manual thematic coding in the first phase. In the second phase, by axial coding, I divided the findings into the separate categories by simplifying some of the findings and combining similar statements, experiences and opinions.

Chapter six summarizes the findings into a table. The table follows the principles of qualitative research techniques by combining and categorizing data (Hirsjärvi & Hurme, 2008). Due to the limitations of this study, the detailed findings and descriptions behind the subcategories cannot be included in the table. Yet, some of the original expressions are presented in the illustrative examples during the analysis in chapter five.

Since the interviews were semi-structured the questions were based on Kanban's theory. The questions were meant to support and guide the conversation rather than to lead it. All questions except for one were not intentionally looking for pitfalls but for differences between Kanban in theory and Kanban in reality. Figuring out why Kanban induced benefits do not manifest is a path to finding features that contribute to unsuccessful implementations.

The background of the informants is presented in the following section since it provides insight to how they have come to their conclusions and how their opinions have developed.

4.2.2 INTERVIEWEES

Michael Holler, Agile Coach at Reaktor

Michael Holler studied in the University of Helsinki majoring in mathematics and computer science. He has 15 years of experience in software development projects varying between 4-man start-ups and 250 person organizations. The development practices of the projects ranged from Waterfall Model through slightly iterative models into agile methodologies. Holler is currently working at Reaktor as an agile coach.

Sami Lilja, Certified Kanban & Scrum Trainer at Reaktor

Sami Lilja is a Master of Science graduated from the University of Helsinki majoring in computer science. He has been working in software development over 20 years and has experience in all the aspects of the software development chain. He worked as the Head of Department for 3G development at Nokia Siemens Networks. During the early stages of his career he developed software with no particular model as the Waterfall Model arrived, later moving on to agile methods. Today, Lilja is working for Reaktor, couching internal teams and working as a consultant, training Finland's biggest companies in software development. He is one of Finland's few licensed trainers by Scrum Alliance[8].

Panu Liira, Principal Consultant at Reaktor

Panu Liira graduated from the University of Technology with a Master's degree in computer science with an emphasis on software business and software methodologies. He has worked over 15 years as a project leader in countless software projects and has thus collected a vast understanding of the field. Today, Liira works as a consultant for agile methods at Reaktor. Before Reaktor he worked in a Start Up, which developed software for big US construction companies.

Juha Vakkila, Director of Online Development at Fonecta

Juha Vakkila has 20 years of experience working in the field of software. With his engineering background, Vakkila has worked mostly in digital services and media environment. He was a consultant for 10 years after which he jumped off to run a tech Start

[8] Scrum Alliance is the largest, most established, influential professional membership organization in the Agile world. (Retrieved from Scrumalliance.org, 19th November)

Up. For the five years, he has been working for Fonecta, one of Finland's biggest search services. During his career Vakkila has encountered countless software development methodologies ranging from anonymous to lean approaches including Scrum and Kanban. Because Fonecta outsources certain software development functions, Vakkila is responsible for the competitive bidding and developer selection of the Online Development Department.

Toni Strandell, Senior Software Architect at Reaktor

Toni Strandell is a Master of Science graduated from the University of Helsinki majoring in computer science. He has 15 years of experience in developing software. First, he worked as a research assistant at the University of Helsinki after which he moved on to work at the Nokia Research Center for six years. After Nokia he joined Reaktor where he currently is working as a senior software architect. In contrast to other Reaktor informants Strandell is the only one currently working as an actual member of a software team instead of as an agile trainer. He has worked with several software development methods ranging from Nokia's "ad hoc" methods to Scrum, and further on to Kanban during the recent years.

4.3 QUALITY OF THE RESEARCH

Quality of a research refers to reliability and validity of the research. In method literature the concept of reliability refers to the repetitiveness of the results. Validity, on the other hand, describes how well the study investigates what it claims to be investigating. How to measure these two attributes has no specific set of rules (Tuomi & Sarajärvi, 2006). For quantitative studies measuring the reliability and validity is relatively easy since everything is quantified. Therefore quantified studies can be tested for repeatability and method's ability to measure what it is supposed to measure with numbers (Hirsjärvi et al. 2009, Uusitalo 1991). For qualitative research however, measuring quality is quite complicated. Due to this fact, the use of validity and reliability has been rather neglected in qualitative research. Still, Hirsjärvi (2009) argues that reliability and validity should be present and measured regardless of the type of research. Hirsjärvi elaborates that a way to increase research reliability and validity is to describe in detail how the study was conducted. In addition, supporting Koskinen et al.'s (2005) point of view concentrating on a small number of observations also enhances reliability. This study follows the above-mentioned guidelines how to improve quality of the research.

4.4 RESEARCH ETHICS

In common language ethics can be defined as the study of what is good and what is bad. What is right and what is wrong? In research, ethics stand for the same moral message. Research ethics can be defined as the application of moral principles throughout the research during planning, conducting, writing and reporting the results (McNabb, 2002, p. 36). This study follows the ethical code of Eriksson and Kovalainen emphasizing the importance of the relationship between researcher and participants since the research was conducted by personal interviews. According to Eriksson and Kovalainen (2008, p. 64-65) the relationship consists of the following three dimensions: voluntary participation, informed consent, and confidentiality and anonymity when asked for

All of the above mentioned dimensions were cared for. First of all, all of the participants were eager to take part in the test. They were also anxious to share experiences among fellow participants about the potential pitfalls after having their own interview. Second and third dimensions, all the participant gave their consent to use their real name and company information in order to support this study. I am very grateful for that since the credibility and authenticity of this thesis depends greatly on being an in-depth theme research with the industry experts. It also makes the thesis more transparent and open for future research, which support the core values of ethics.

5 ANALYSIS

The purpose of this chapter is to analyze the information derived from the interviews. This thesis uses data driven analysis in order to create a theoretic whole of the qualitative data received through empirical research. According to Miles and Huberman (1984) data driven analysis can be divided into three phases. Similarly, this study follows the three phases in managing the retrieved data.

1) Reduction of the data (simplification)

2) Clustering of the data (categorization)

3) Abstraction of the data (creating theoretic concepts)

To clarify, data driven approach does not equal to amount of data used: it is a method to study a phenomenon that has no previous context (Tuomi & Sarajärvi, 2006). According to Tuomi and Sarajärvi, a data driven study builds its own categories based on the findings. This study builds presents the created categories in the beginning of chapter six. The interviews brought up several challenges that Kanban driven software projects face. By combining the similarities between informants' experiences and reasons behind them the pitfalls could be divided into five main categories, and further into subcategories within the five main categories.

The categories were created based on two factors. First, when or where the pitfall emerges. Second, for which reasons the pitfall emerges. The categories form cohesive wholes, which are strongly related to each other. Some of the categories overlap because reasons behind the pitfalls have several consequences that fit several categories. This said, I found that these five categories were the most pragmatic ones to describe factors influencing the emerge of Kanban's pitfalls.

CATEGORY 1. FALSE INTENSIONS BEHIND THE ADOPTION

CATEGORY 2. CHANGING THE MINDSET

CATEGORY 3. KANBAN IN REALITY

CATEGORY 4. KANBAN IN THEORY

CATEGORY 5. SUITABILITY OF KANBAN

In order to facilitate the comprehension of the analysis, I present the five main categories in the beginning, tying the individual pitfalls into larger categories as the report draws on. It is easier to understand the bigger entities of Kanban's potential pitfalls first and then dive deeper into individual pitfalls that emerge from the different entities.

Before analyzing the categories let us look at the definition of a pitfall. It is essential from the research questions' point of view to understand that the term "pitfall" is not a synonym for incorrect, failure or a mistake. The word pitfall refers to a *"potential trap if not* _insert a verb_*carefully."* By definition, it is "a hidden or unsuspected danger or difficulty". [9] Another translation describes pitfall as "a danger or problem that is hidden or not obvious at first".[10]

In this thesis, a pitfall refers to a challenge that Kanban faces and to actions that nourish Kanban's failure. The potential pitfalls require vigilance from teams that are adopting Kanban. This thesis follows the previously described three-phased data analysis process creating by abstracting the data.

The following five sections will analyze the individual pitfalls by discussing experiences and illustrating them with examples from real life. The analysis discusses the cause and nature of the pitfalls. Should several informants have experienced the same pitfall, the study will presents all the opinions in contrast to each other.

5.1 FALSE INTENSIONS BEHIND THE ADOPTION

During the interviews, one of the themes that kept surfacing was the nature of the motivation of an organization that was keen on adopting Kanban. In several cases, the experts found that organizations chose Kanban with false pretenses targeting their own personal goals. Twisted motivation behind the reasoning is causing difficulties in achieving the original goals of Kanban driven development. Here are three false intentions to adopt Kanban that experts felt were detrimental for project success.

5.1.1 KANBAN IS EASIER THAN SCRUM

[9] (www.oxforddictionaries.com, retrieved 26th September 2014)

[10] (www.merriam-webster.com, retrieved 26th September 2014)

A common cause to switch from Scrum to Kanban is that Kanban is perceived to be easier than Scrum (Kniperg, 2014; Mayer, 2009). What is meant by "easier" is that a development team experiences the Kanban method to be more flexible and doable than Scrum. As explained in the literature review, Scrum entails a list of roles, artifacts and ceremonies with no tolerance for exceptions or modifications to the processes. Therefore Scrum is often experienced too difficult to cope with. Now, because Kanban provides more liberties given neither pre-defined rules nor artifacts, it is common to turn to Kanban when Scrum starts to cause "pain" in the system. Lilja elaborates that pain refers to everything that a team feels to be difficult, unwanted and new. For example, by limiting the WIP a constraint is exposed. The team members working on that particular phase are therefore under pressure from the whole process and can feel underperformed and accused. In this case, no further tasks can be undertaken before the constraint is uplifted causing more pressure to the team than before because nothing is getting trough the system. This can be experienced as a sort of pain, which gradually aims to heal the process. Both Kanban and Scrum have to "break the bone" and set it straight before it can heal properly.

Furthermore, the Kanban Method is intuitive to understand and it gives rather free hands to the development team to plan their work. The only rules concern the workflow visualization and work in progress. Actually, there are no WIP limitation rules either - only a recommendation that WIP should be decreased. Lilja explains that the idea of Kanban with its freedom of hard limits and restrictions seems luring to many companies. Especially in contrast to Scrum. Lilja has dealt with numerous companies that ask for help when Kanban fails to fulfill their expectations. After a short analysis, he has been able to identify a pattern: from Waterfall to Scrum and from Scrum to Kanban. The shift from Waterfall to Scrum has caused too much pain, which has led to the adoption of Kanban. However, this is the decisive turn where things go wrong and the pitfall becomes visible. In essence both Scrum and Kanban will create pressure and cause pain to the process when applied correctly. Scrum in other ways than Kanban. The point is that with Kanban it is easier to avoid the pain. By not limiting the WIP, which exposes the bottlenecks, improvement cannot take place. This induces a so-called superficial implementation, which will be further discussed later in this chapter. Hence, the false intention behind the adoption of Kanban is usually a decisive factor leading to manipulated implementations from the very beginning. Besides, Strandell states that it is pointless to compare the "easiness" of Kanban to Scrum. All the experts agree on the fact that these two software development methods are concepts of different level. This is a

commonly acknowledged fact by agile communities that many uninformed people unfortunately get wrong at the first glance (Mayer, 2009a; Anderson, 2010). Lilja put it this way: "It is like comparing coconuts to cross country-skiing. I am not saying that there is anything wrong with that. Just that the results might lead you off track." Most importantly, the methods are not mutually exclusive, so the shift from Scrum to Kanban is pointless. It is actually quite common to manage projects with both Scrum and Kanban.

To summarize, software development teams have shifted from Scrum to Kanban because Kanban seems more "easy-going" and flexible than Scrum. Supporting this act, Kanban is easier to misuse because it lacks detailed guidelines. The adoption of Kanban to avoid pain and pressure will most likely induce a superficial implementation, skirting Kanban's actual value proposition.

5.1.2 Intrinsic value of Statistical Process Control

Statistical Process Control uses control charts among other mathematical tools to study variance and other key metrics of a process. During the length of his career, Lilja has experienced several cases where managers had adopted Kanban due to this precise feature. As a result, managers were now asking Lilja what was wrong with the process. They had done everything by the book and the process was still unstable and unpredictable. Lilja recognizes a pitfall in considering SPC as an intrinsic value of Kanban. In his opinion SPC should not be considered as the most essential feature of Kanban. SPC should be considered as a side product of measuring lead-times, but that the core value of Kanban lies completely someplace else. Kanban's focus should be on optimizing the value stream and work flow – not on specific numbers that managers can monitor through the charts. However, managers tend to love numbers because data can be used to make predictions and estimates for their future strategy and advocate their desired budget. Although Kanban does produce numbers, the numbers should not be the primary purpose of adopting Kanban Method. Lilja fears that the numbers shift the focus to the future whereas Kanban should focus on the situation here and now. In a situation where the demand is purely value driven the tasks vary so much that the mathematical models start to fall apart since the average lead-times do not match the sheets. Instead of making the obvious assumption based on numbers that something is not right, one should not look at the numbers but follow lean values and focus on the flow.

Shifting the focus to command and control-based data can in the worst-case steer the whole process off-track based on manipulated numbers. This is where many processes step into a deep pitfall. Lilja concludes that the over mathematical approach to Kanban should be secondary to optimizing the flow and overall performance of the process. Supporting Lilja's view, Latzko (1995) stated that paying more attention to variation reduction and searching for special and common causes shifts the focus away from the essential leading the process to sidetrack.

5.1.3 KANBAN IS TRENDY

Agile methods have been proven to be more productive and cost-efficient than its' predecessors fixed budget, schedule and scope approach. At the moment, Kanban has a reputation as one of the trendiest software development frameworks that is built on agile- and lean values. Vakkila recalls several incidents, where he as the customer, has encountered consultants using Kanban with no pragmatic justifications what so ever. When he challenged the consultants to explain why this particular method was chosen for this specific project, the answers were vague and unrelated to the project. In Vakkila's opinion one should always questions why something is done in a certain way and "not to do it, only for the sake of doing it". If Kanban can be justified he is more than happy to support the approach. On the opposite if Kanban is brought up just for the sake of "name-dropping" to sound professional it is a definite no-go for him. From an academic perspective, this might seem trivial. Still, the unfortunate truth is that Kanban is often not adopted for its original purpose but to appear "trendy" in the eyes of the customer. Vakkila challenges the software development teams to think whether Kanban is the best option.

Vakkila faced the pitfall from a customer's point of view whereas Strandell, as a member of a software team, has been demanded to use Kanban by the customer. He shared a similar experience while conducting a 6 MEUR project for the Finnish Government. The stakes were high and the PO was one of the customers. The PO had been given the instructions to use Kanban. What actually happened was that the PO had a backlog filled with tasks enough to last for a year. He was very proud of it since now he could estimate when everything was going to be ready. He was able to display all this information to his boss, who was very pleased with the project. What the PO was actually doing is Waterfall disguised as Kanban.

The method had nothing to do with agile software development. Strandell explains that this is something he comes across every now and then. In this case, the customer had not understood the pure essence of Kanban and how it creates value. Listing tasks to last a year destroys the core idea of Kanban as an iterative agile framework and prevents re-prioritizing the tasks to support the constantly changing environment. Strandell later found out that the instructions came from a party with no knowledge of software development. Kanban was simply demanded because it was perceived as the coolest new toy on the market.

To adopt Kanban just because it is trendy or the newest fad often leads to difficulties. If one is not willing to commit to the changes that Kanban initiates to improve a process, there is practically no difference to Waterfall. What happens is that Kanban is integrated into an existing e.g. Waterfall Model, which is possible, and nothing changes. Strandell sais that the only common factor between Kanban and the customer's method was "colorful post-its on the wall."

5.2 CHANGING THE MINDSET

Changing the mindset is an umbrella term covering all the five main categories. It is the red thread connecting all the pitfalls: all the failures ultimately come down to erroneous mindset towards Kanban. For example, the false intentions behind Kanban's adoption are directly linked to having a misled perception of what Kanban is. Still, I have chosen to present the pitfalls of changing the mindset separately since there are some that are purely attitude and mindset related. Yet, it is important to note that changing the mindset is crucial for all five categories.

5.2.1 LACK OF COMMUNICATION AND SHARED GOALS

A big part of the development work is actually communication, which measures ones social skills. Strandell made an interesting remark on how technical issues are trivial in his opinion. He elaborated the comment by stating that "everyone can make software, the decisive point is how efficiently one can do it." The point is that even though technicalities can be challenging and very difficult to crack, bu t the most difficult part is the "people stuff" as he calls it. In the terms of Kanban, the bottleneck of the process is communication and the ineptitude to agree

on the goals of the project. He argues that social issues contribute to the greatest percent of why projects fail. If people fail to communicate properly, no process management tool can rescue the situation, not even Kanban. No method can guarantee success.

The reason why communication is so important is that each process has to have a goal and that the goal has to be agreed upon. Strandell argues that the most important single factor to support process improvement is to share a common goal. The goal does not have to be strictly a defined product a year from now, but a shared understanding of what the project aims to achieve. Kanban supports the use of milestones, which are staging posts to reflect and contemplate the future tasks. Teams should ask themselves whether they are still on the right path or has the situation changed? The milestones lead the way to the bigger target and are created together in order share understanding of the bigger goal. Milestones also help to comprehend the scope of the project and the amount of work the team faces. They lead the way to the ultimate goal and create a target path that shapes it form along the way. Agile milestones give the opportunity to shape the long distance goal when a project moves on and more knowledge is attained about true needs instead of "just-in-case" features. Strandell shares the view of ever-changing target and target path. The ultimate goal will never be there where it was first imagined to be because environment is constantly changing. Agile methodologies were created to respond to this need.

Strandell continues that the reason why discussion about the goals is crucial is because without having a shared vision and explicit understanding of how things work and how work is actually done, any discussion of problems tends to be emotional, anecdotal and subjective. When everyone shares the vision of the goal the understanding of what needs to be done takes a more rational approach. Only then can the team begin to make unanimous decisions regarding the changes in the system. The choices will be more rational and there can be an objective discussion of issues facilitating consensus around the improvement suggestions. Moreover, sharing goals clarify and make the distinctions between value adding and wasteful activities easier. If each person would have his or her own personal goals, then the discussion of what is waste would be completely subjective: an activity for one person might add value and for another person it might be a complete waste of time.

In order to create a shared vision, the milestones and requires tasks have to be defined. Strandell acknowledges that it is demanding since there tend to be as many opinions about the possible backlog as there are people in the project. A good place to start is to understand the

basic need of the customer. Even though Kanban supports Strandell's approach navigating the target path, one has to be extra careful not to rely completely on Kanban but to actively pursue discussion.

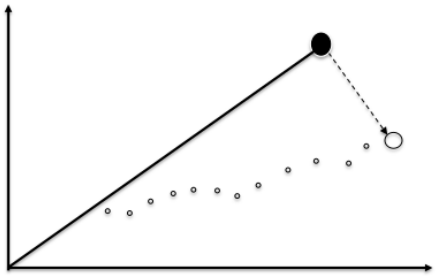

Figure 10 A simplified illustration of Kanban's target path based on drawings by Strandell.

Figure 10 illustrates how Kanban is better equipped for a changing environment. The black line presents the traditional fixed scope Model striving towards its goal, the black circle. However, due to ever-changing circumstances such as technical evolution of some sort or a change in customer demand shifts the goal. Let us assume that the goal has moved to where the white circle is now. The small white dots illustrate the milestones after the goal has shifted and the route is recreated. With Kanban the development team is able to meet the more accurate need of the customer that changed during the process. The milestones move along the long distance goal, they explore the route. It is likely that the ultimate goal changes several times before reaching the target. With Waterfall the shift is impossible. The need for rethinking and re-prioritizing the tasks might even have gone unnoticed.

As explained in the literature review, Kanban is built on short iterations supporting small feedback loops enabling quick and agile modifications in the project. Honkonen (2014a) described the situation as follows: "You are standing at a beach. There is a black see ahead and you cannot see the bottom. You suspect that the bottom is rocky but you do not know where the sharp rocks are. What do you do? Do you run straight ahead without any hesitation or do you walk slowly into the sea exploring the sea with your feet, avoiding sharp rocks and trying to find a stable hold to proceed?" It is the same way with software. This is the value that Kanban has to offer but many fail to grasp it due inadequate communication. Strandell concludes that Kanban is a framework to support the change within the process - not to make it.

5.2.2 CUSTOMER RESISTANCE

Even though Kanban is designed to minimize resistance to change by starting with the existing process, it still experiences customer resistance. The reasons behind the resistance lie in the existing nature of software development.

The Waterfall Model provided customers with exact delivery dates and features. The budget was fixed and schedules were rock solid. From customer's point of view, buying software was in many ways easier or at least more straightforward. This arrangement created an illusion that everything was in control and that the customer was getting what he wanted. Customer ordered a product X, team delivered product X, most probably late and over budget but still. Supportively, Addison and Vallabh (2002) have identified unrealistic budgets and schedules to be in the top two most frequent risk factors in software projects. In a mature Kanban process, the smooth task flow and short feedback loops enable the schedule and budget to be estimated based on facts, not on guesses. What Kanban proposes is a product X^2 but it does not make any promises about included features. For this reason agile methods are more difficult and time consuming from the perspective of customer even though the results exceed Waterfall in every aspect.

Liira has fought against these negative associations with agile countless times during his career. In his opinion, it is a rule rather than an exception for the customer to favor fixed scope methods, especially during contract negotiations. In a typical situation, a customer has a clearly defined picture in his mind of what he wants. Waterfall promises him just that. Instead, Kanban allows modification of the backlog even after the process has started. Liira points out that this should be a good thing but for many customers this feature of Kanban has been troublesome. Customers often fail to understand the changing nature of the work even at their own expense. The arrangement is beneficial for both parties but the unfixed budget scares many product owners and customers. Liira has had to explain several times that the outcome is still going to be the best possible: the software team is always working on the most important tasks, thanks to backlog prioritization that contributes to the best possible outcome. Holler shares Liira's point of view by stating that it is more common to hear prompt demands from the customer than to have an open contract. Yet, he also understands the customer's point of view. Agile contracts require more trust from both parties.

All the experts acknowledge the amount of mindset change agile requires from the customer. However, very often the resistance to accept Kanban's principle not to promise hard facts

increases resistance creating the difference between successful and unsuccessful project. Therefore, customer resistance can be defined as one of the major pitfalls of Kanban. Although it is not an in-build feature of Kanban it is a pitfall to be associated with it, which causes projects to fail. What counts is that while adopting Kanban special attention should be paid on creating trust between the customer and development team.

5.2.3 Team and Management Resistance

Following the previous topic, team resistance is a common pitfall of Kanban as well. Lilja described the adoption of Kanban from the team's perspective as a leap of faith: very promising outcomes, but no certainty to reach them. Kanban changes the work environment by increasing collaboration. Visualization also necessitates the team to talk to each other and forces them to work together. Lilja embraces increased communication as one of Kanban's major contributions. Nevertheless, it is easier said than done: Kanban does not come natural to everyone. Collaboration and communication for a software team that is used to keeping to themselves might in reality turn out to be very hard. Therefore it has to be a conscious decision to strive towards Kaizen and to be willing to make the change.

A team member and management might also resist the change due to Kanbans non-hierarchical nature. Every member of the team is equal and has a say in the backlog. For a team member with a senior status this is sometimes hard to accept. Lilja illustrates the magnitude of change by explaining a true scenario where a developer had worked almost a decade to be promoted into a senior developer position with more rights and command power during the Waterfall Model. Now, with Kanban, all the rights and liberties are secondary if not completely void. It is not an easy equation to cope with. Either way this is an issue that should not be underestimated and left to turn into a pitfall.

Another point of view is when management accepts the team decision to make software with Kanban but is still expecting quarterly reports and fixed budget presentations – all characteristics of the outdated predecessor. Both strategic level and operational level have to match as well as thinking has to match actions. Thinking creates the system and the system is the foundation for process performance. Logical explanation for unsuccessful Kanban project root to not supported agile thinking. Even though the Kanban System is in place, Kaizen is missing. To change the mindset teams have to start questioning issues that are working and

issues that are not working. In other words, to adopt Kanban the change has to start with agile thinking. The focus should be on how thinking can be improved. After that one can concentrate on improving the system and performance.

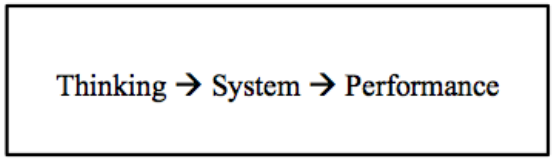

FIGURE 11 Lilja's comprehension of how thinking affects the system and how system further affects the performance.

5.3 KANBAN IN REALITY

This chapter discusses the potential pitfalls of Kanban in action. Are there elements in Kanban theory that the informants felt were unrealistic in real life software development? Here are two potential pitfalls.

5.3.1 MISSING WARNING SIGNAL

Lilja argues that one of Kanban's deficiencies is the lack of a foolproof warning signal. Lilja has met several companies with similar problems calling him for help where everything on the surface seems the way it should but Kanban makes no progress. Everything in the process appears to be in control and follow Kanbans core values. It requires a lot of talent to spot where things go wrong. Many times, Lilja has wondered if there would be any easy checklist for Kanban System to reinforce it. The lack of a warning signal means that companies and teams can go on for years without noticing that the system is not working properly.

In comparison to the Waterfall Model, Lilja explained the cost of failure in the graph below. The graph demonstrates the relationship between cost and time in software development, for both long feedback loop project and short feedback loop project. The graph illustrates how defects that are noticed sooner are less expensive than defects noticed later in the process. Moreover, the relation is not linear. It is significantly more expensive to discover bugs later in the development chain than to detect them already in the beginning. The cost of long feedback loop mistakes emphasizes the importance of spotting the mistakes before the final product reaches the customer. Slaughter et al. (1998) studied the cost of failure in software

development during the development process supporting Lilja's argumentation. They made similar findings where the cost of failure ascended intensely towards the end. Thereby, Lilja recognizes a major pitfall in Kanban's detection system, which would warn the team that something is wrong.

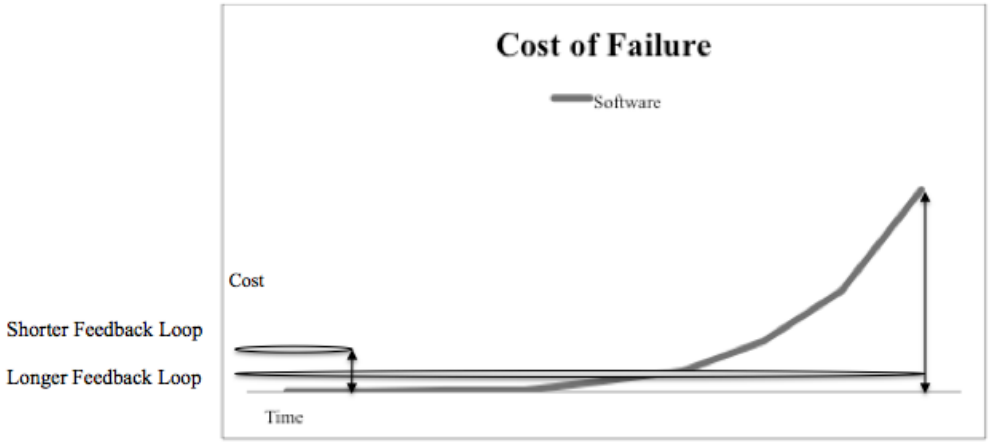

FIGURE 12 Cost of Failure based on drawings by Lilja

4.3.2 SUPERFICIAL IMPLEMENTATION

Superficial implementation of Kanban System was the only pitfall, which all five experts denoted during the interviews. Therefore this thesis considers the superficial implementation as a matter of great importance and as one of the most significant findings of the study. The discussion of a superficial implementation of Kanban is interesting since it forces one to draw a line between Kanban and Kanban imitation. The way Kanban was superficially implemented took different forms depending on the informant.

Holler elaborates that development teams are usually eager to adapt the first three core properties of Kanban and leaving the rest two to faith. Teams are visualizing the workflow, limiting and managing the WIP but making processes explicit and improving collaboratively has gained less appreciation among the teams resulting in somewhat abstruse agile structures. Leaving out core properties is one way of implementing Kanban on a superficial level.

Complicating things even further - Kanban is supposed to be tailored. As a result, because everything is tailored it is difficult to detect, which decisions were made to support the process and which were made at Kanban's expense. For example, Lilja and Liira have experienced numerous teams claiming to be using Kanban but after exploring the Kanban

board, they have come to realize that the board had nothing to do with the real workflow in the system. And if the board does not reflect the reality, how can proper decisions about future tasks be made? If a bottleneck is not visible on the board, team members cannot react and help sort the bottleneck out. They will just go on building some new features overcrowding the testing and worsening the situation all due to inaccurate Kanban board visualization. In many cases, the board is visualizing a dream workflow; one that the team and managers want to be true. This is the very first pitfall. The only way the visualization is going expose the bottlenecks is by presenting the reality. By leaving phases out or miss presenting them, bottlenecks potentially fall off the grid making the flow even more complicated to understand. Holler proposes that one of the reasons that visualization is not up-to-date is lack of discipline. Liira supports the view by exemplifying leaving dishes on the table instead of putting them into the sink or washing machine: it is natural for human beings.

Strandell thinks that the reason why the board does not match the reality or is inaccurate leaving phases out etc. is because people are ashamed of some phases, which they want to hide. A developer might want to hide the fact that he is idle and that there is slack in the process. All three experts conclude that the only way the board can work is by visualizing the truthful situation of the process including all the phases the task has to go through. Lilja adds that the easiest way to get an overview of the board and process is to observe the team in action and question the lead-times and defect rates exposing the real nature of the process or an imitation of a Kanban board.

Vakkila instigates to follow the core properties in order to implement Kanban correctly. Nonetheless if the willingness to change the organization is low, the implementation of Kanban can stay very shallow. Liira notes that there is no pain, but there is also no gain. Nobody hurts, but nobody wins. In the worst case, as discussed in the previous section, the teams do not even know they are not doing Kanban at all but just fancy post-it illustration. A Waterfall Model visualized with kanbans is not Kanban. It requires discipline not falling back to Waterfall and the team has to slice the task to be small enough.

Liira compares the phenomenon of superficial Kanban implementation to “ScrumBut”. According to Liira, one has to have an extra good reason or a something that covers the “but” part in order to differ from the traditional behavior. It is very likely that if one makes alternations to Kanban guidelines it is to cut corners. To conclude, the potential pitfall of superficial implementation is so well hidden that it might never be encountered.

5.4 KANBAN IN THEORY

The fourth category presents potential pitfalls in the theory of Kanban, which is a proxy of existing theories and Anderson own additions as explained in the literature review. The experts provided insightful views to the theory and potential challenges that it includes.

5.4.1 LACK OF DETAILED GUIDELINES

Holler has a contradictory view on the theory of Kanban. On one side he feels that the greatest strength of Kanban is customization. Everything can be tailored to the process needs. On the other side he admits that the lack of detailed guidelines is one of Kanban's biggest challenges and reason for project failures. Although Kanban's theory is vast and draws on several different disciplines, the theory is not addressing some of the fundamental features of software development. Holler gives a few examples such as the team size; in his opinion teams are best kept small, 4-5 members. Kanban says nothing about the team size nor does it address any of the technical issues regarding testing, emerging architecture, development tools or pair programming. There should be a stable cadence delivering the software but no guidelines how this should be organized is provided either. Moreover, customer collaboration, which plays a crucial role minimizing the customer resistance discussed earlier in the text has no in-built features in Kanban. Similarly, Liira finds it especially challenging that Kanban has no built-in meet-ups especially retrospectives like Scrum. According to Liira, the lack of mandatory meetings is a challenge because the team itself needs to realize they need them. Speaking from experience, Liira notes that the most needed retrospectives are the ones when the teams do not know they need them. Often they are held too late, when the pain is already existent.

Holler concludes that there is actually nothing wrong with the theory of Kanban itself. The lack of guidelines only becomes a pitfall if Kanban is thought to be sufficient by itself. Holler recalls the theory of Kanban and sites that "Kanban is a framework which can be integrated into an existing process management tool." Now, if that is the case, the process should be fine given that the tool is not Waterfall. Sharing Hollers point of view, Liira describes Kanban as a very powerful change management method which by itself is inadequate to manage the development process. Again, the pitfall is not the theory of Kanban itself, but poor

interpretation of it. Despite all the above-considered matters the theory presents a hidden pitfall by so many by assuming Kanban to be sufficient by itself.

Lilja approached the topic from another point of view. What was buzzing him was the nature of Kanban's core properties and principles. Some of them are overly simplified without any guidelines how to proceed with the suggestion. Lilja questions how naturally Kanban's guidelines can be put to action. He uses the fact how Kanban encourages teams and customers to communicate as an example: but how? But when? The theory is blunt and frankly a bit shallow. Lilja does agree that communication is important and a nice thought but challenges the action-ability of Kanban's principles. Both Holler and Lilja felt that Scrum provided plenty of detailed guidelines giving more to chew on than just the plate. Again, Kanban requires an existing process management tool and is not sufficient by itself. Lilja expands his view of shallow guidelines by painting the bigger picture. All of the pitfalls are connected. Due to undetailed guidelines, he felt that many Kanban implementations remained shallow, which is a symptom discussed in the previous section.

Holler, Lilja and Liira all felt that the theory of Kanban is problematic comparing it to a double-edged sword. On one side, the lack of detailed guidelines and rules can contribute to undisciplined process control. On the other side, Kanban gives the liberty to self-design the process and needed artefacts to its needs by minimizing the resistance to change. Balancing between these two is hard and where the balance is found, is context depended. Liira quips that there are no best practices in software development other than to think.

5.4.2 Dissonant concept of Waste

Another issue that two of the experts consider problematic is the concept of waste. First of all, Holler does agree that software development has many wasteful activities such activities supporting IT infrastructure, which include heavy processes, enormous amount of testing, loads of documentation and technical releases. Apart from these activities lean thinking also considers everything that is not directly contributing to the product as waste. Therefore, Kanban considers many tasks that are in our experts' opinion crucial, waste.

Strandell has difficulties accepting the lean approach. He has found that activities not directly adding value to the product have been the most rewarding leading to successful outcomes. He disagrees on the concept of value-adding tasks. For example, Strandell notes that thinking in

itself does not fill the requirements of a value-adding task. Should it be considered as waste? Planning is a form of thinking, at which point does thinking and planning turn into waste? "Planning six months ahead, yes maybe, but the next week, no", Strandell thinks out loud. Only coding could in this case be considered as directly value-adding task. However, stopping for a second and looking up from the keyboard can improve the process by steepening the tangent. Concentrating on the effectiveness instead of efficiency just to clarify the goal is healthy. Strandell does not claim that Anderson himself perceives thinking as waste. However, the concept of waste in some cases has gotten a peculiar form in the minds of the customer because of the dissonant explanation. Sometimes he feels that the customer expects the team just to code without any planning at all. Strandell finds that for a customer, it is hard to understand that sometimes it is better to do nothing and stop for a minute and think. The common perception, however, is that everybody has to have something to do. The concept of waste is therefore misleading. Waste is not black and white; customer should definitely watch this pitfall since a huge part of coding is finding the right target path. Strandell continues that is better to do right things slowly than wrong things fast. He admits that he feels strongly about this particular concept. Especially because during his career he has had to face several situations where customers have had a twisted perception of value adding tasks. Hearing comments such as "It is nice to see that everyone has had their fingers on the key board this week" are very common. Instead, the emphasis should be more on figuring out the right things to do, searching for the right goal. It is not that customers do not value thinking but very often customers favor coding over thinking. This is a problem. Strandell has a mantra "Anything you can leave, leave it. Don't code it." The reason is very simple. Everything the software team produces, codes, makes the product more complicated. What compensates for this is the value that the code creates for the product. If the team can produce the same value with less code, it is relatively more valuable. To summarize, the concept of waste is dissonant causing customers to perceive some crucial value-adding task as secondary to coding.

5.4.3 SELF-DELIBERATE PROCESS

Kanban becoming self-deliberate is a higher level concept compared to the previous topics. Kanban becoming self-deliberate means that one is doing Kanban for the sake of doing Kanban, being blinded what the actual goal is. After implementing Kanban, making incremental changes and modifications, the WIP limitation becomes a priority, as it should.

By focusing on the project from the technical point of view can induce a situation where the process is improved continuously. The pitfall is to not look at the big picture. Is this the process that we should be improving? Is there a better way to reach the goal? Tuning up the process to eternity can easily get an upper hand. For a team with background in engineer this is a very probable scenario Holler adds. One of agile manifestos principles is "individuals over processes and tools". Teams should focus rather on relationships rather on the process. The process performance efficiency level is trivial if the team strives towards an expired goal. The only way to prevent the process from becoming self-deliberate is to focus on relationships and talk to the customer. Holler comments that in the end it is all about the people: making software happen is about relationships, about discussion, about having common goals.

5.5 SUITABILITY OF KANBAN

Kanban squeezes out mediocrity and requires high-performance causing pain. Hierarchical bureaucracy breeds incompetence and feeds off mediocrity: the organization performs accordingly. Faced with the choice between high-performance and the mediocrity, traditional management opts for mediocrity. – Denning, 2012

In 2012 Steve Denning wrote an article "The Case Against Agile: Ten Perennial Management Objections" which presents ten excuses that big corporations use not to adopt agile. The reasons varied from "Agile only works for small projects and our projects are big" to "Agile lacks project management processes". What is interesting is that all the informants were of the opinion that agile does work for big corporations and shared Denning's opinion of Kanban's suitability. However, Lilja and Vakkila presented two scenarios where the suitability of Kanban is questionable.

5.5.1 USING KANBAN FOR PURE VALUE- OR FAILURE DEMAND

The situation for which Kanban is suited for depends on the demand in question. Lilja explains that demand can be divided into two categories, which define the method most suitable to match the demand. There is value demand and failure demand. Value demand is

real demand that creates value for the customer. Failure demand is everything else needed to be done but does not contribute to real value. Failure demand comes from poor value creation when something has gone wrong in the value creation process. Embodiments are bugs in the code, poor usability, missing feature etc. All these create failure demand. The failure demand creates waste, which lean thinking tries to eliminate. For example, help desk workers answer to failure demand. When a team starts a new project with no burden from previous code or maintenance issues, it is pure value demand. The situation is commonly referred to as the green field situation.

In Lilja's opinion, Kanban is not best suited for pure value demand, the green field situation. In his opinion Kanban is at its best in a situation, which has both value and failure demand. Lilja justifies his point of view with his own experience in the software field. For a pure value demand situation Lilja would use Scrum, which has a strict set of rules boosting team performance to the fullest from day one. The loosely built Kanban however, does not force the efficiency to the top level immediately. Nor does it provide any specific guidelines how the process should be managed. For an older project, that has burden and countless rows of code, maintenance is important. Scrum starts to fall apart when maintenance and unexpected expedite request emerge because failure demand increases. The failure demand increases because the existing architecture requires work.

Very often Kanban is used to handle demand that it was not designed for. For example, improving helpdesk work by improving the pace of by which incoming calls are answered does not support the goal. Failure demand is still failure demand and coping with failure demand faster does not give you what you do want: an improved process generating better products in the first place.

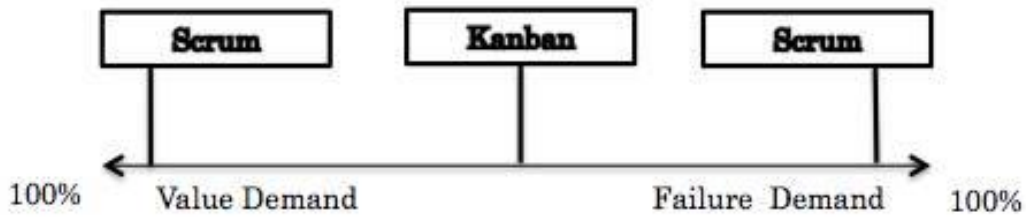

FIGURE 13 Demand based process approach based on drawings by Lilja.

5.5.2 THE METHOD IS TOO HEAVY

In accordance with the literature propositions Vakkila states that everything comes down to the nature of the software. Similar to Hawrysh and Ruprecht (2000) who state that a single methodology cannot work for the whole spectrum of different projects Vakkila argues that Kanban does not fit all purposes. He makes the distinction of the demand, not between value and failure, but between "severity" and "non-severity". He exemplified his opinion by illustrating severe software projects. When developing software for a bank, NASA or a heart surgeon it is obviously a top priority that the software does not contain any defects. Updates for the software are hard to implement and flaws in the code lead to serious consequences. Yet, when his team at Fonecta is developing software the consequences of "showing the wrong number" are not that devastating. Therefore, he favors software that has not that much "luggage". In Vakkila's opinion if the severity of the software is low and the projects small it is a waste of time to use Kanban. In his opinion moving post-its on the wall if there are only two team members is nonsense. They can discuss the matter and express what they are doing face to face. Vakkila does agree that Kanban has its perks but in for him personally the framework is just too heavy. He has experienced faster lead-times with no particular method when his teams have focused only on the code. If the tasks are small enough and the crew is competent, visualizing a Kanban System and following all the five core properties can be considered waste. For that reason Vakkila does not like the idea of making processes explicit. For him making something explicit is extra work because the next task or project is going to be different and then processes should be redefined and made explicit again. Following rules for the sake of following rules makes the process self-deliberate.

Vakkila's motto for software development is "Get things done. Get it released, that's all." For him Kanban has turned into waste due to the nature of the software. Vakkila suggest that product owners should gather small teams and just "start doing it". With experienced people it should be possible. He acknowledges that his method is likely to increase the amount of defects but the decreased average lead-time compensates for that. He admits that with his method rockets would probably fall off of the sky but if it is only phone numbers that one is concerned about Kanban too heavy. He concludes that it is absurd to shift pink post-its back and forth between two team members.

6 FINDINGS AND DISCUSSION

The primary purpose of this chapter is to summarize the findings relevant to the research question. *What are the hidden pitfalls of Kanban in software development?*

The secondary research question supports the first question by specifying the cause. *What are the reasons behind the pitfalls?*

In this qualitative study, the nature of the pitfalls and reasons behind them are piquant. The study was able to identify five categories of potential pitfalls that were further divided into subcategories. Moreover, this chapter discusses some of the most essential features of the pitfalls in contrast to previous research on how and why agile fails.

TABLE 2 Summary of the Findings. The table summarizes the findings of the study allocating individual pitfalls into main categories. The reference made by an informant is listed in the column next to the categories.

CATEGORIES	SUBCATEGORIES	REFERENCE
1. CATEGORY FALSE INTENSIONS BEHIND THE ADOPTION	KANBAN IS EASIER THAN SCRUM	LILJA & STRANDELL
	INTRINSIC VALUE OF STATISTICAL PROCESS CONTROL	LILJA
	KANBAN IS TRENDY	VAKKILA & STRANDELL
2. CATEGORY CHANGING THE MINDSET	INCORRECT FOCUS AND MISSING COMMON GOALS	STRANDELL
	CUSTOMER RESISTANCE	HOLLER & LIIRA
	TEAM AND MANAGEMENT RESISTANCE	LILJA & STRANDELL
3. CATEGORY KANBAN IN REALITY	MISSING WARNING SIGNAL	LILJA
	SUPERFICIAL IMPLEMENTATION	HOLLER, LIIRA, LILJA, VAKKILA & STRANDELL
4. CATEGORY KANBAN IN THEORY	LACK OF DETAILED GUIDELINES	HOLLER, LILJA & LIIRA
	DISSONANT CONCEPT OF WASTE	HOLLER & STRANDELL
	SELF-DELIBERATE PROCESS	HOLLER
5. CATEGORY SUITABILITY OF KANBAN	USING KANBAN FOR PURE VALUE- OR FAILURE DEMAND	LILJA
	METHOD IS TOO HEAVY	VAKKILA

Category 1 presents pitfalls that came to prominence when a team or management had false intentions to adopt of Kanban. The informants had experienced situations where Kanban had been adopted because it is one of the most popular topics in agile software development at the moment. Trapani writes in his post that sure agile is trendy but what good does agile do if it fails for the specific goals and business. Being in "the in crowd" then will not matter, will it? Cohn (2008) addresses the same issue in his post about "How to fail with agile" when teams and management adopt Scrum for the sake of Scrum. The pitfall is to adapt Kanban without further analysis how it is going to benefit the project. Name-dropping Kanban is similar to what has happened with Scrum in the past.

Another subcategory of false intentions is to consider Statistical Process Control to be an intrinsic value of Kanban. Similarly, Chapman and Powell (2014) identified "Trying to estimate velocity perfectly" as one of the pitfalls while adopting agile methodologies. Focusing on the mathematical nuances of Kanban is leading the focus off the most important issues. Chapman and Powell continue that teams spending more time estimating and planning than developing worse than having few defects in the code.

Category 2 demonstrates how the change in mindset affects each particle of Kanban implementation. The importance of being able to change mindset to support agile and lean values is crucial since it influences each of the five categories. Without a shift in mindset the likelihood of other pitfalls to emerge sky rockets. According to a Forrester survey, companies listed "the change in business culture" as the most difficult and important task during an agile implementation (Ambler, 2014). Both Schwaber (2011) and Mayer (2009a) have recognized the magnitude of agile mindset as well. They state that organizations fail to adopt Scrum because they mistake Scrum just for a list of practices. Cohn (2009) continues that the challenges that Scrum faces lie deeper on the strategic level since Scrum involves fundamental transformation in the way work is managed. The challenge is to get managers to think agile. If Scrum is integrated into an existing process in traditional management surroundings with hierarchical bureaucracy, it is bound to be challenging. Both Scrum and Kanban require a mindset that teams have trouble adjusting to.

Four out of five informants listed customer, team and management resistance as potential pitfalls of Kanban. Brown (2012) has written several posts why in his opinion agile fails. He has come to the conclusion that agile fails whenever it is forced on a team or an organization. Being accustomed to another method and shifting to agile is not easy (Brown, 2012).

Developing software in a certain way for years and running a smooth transition is just not doable in his opinion. There is bound to be resistance. Succeeding, Bar-Nahor (2014) and Cohn (2008) report resistance to change as a primary factor causing Scrum to fail. The only difference with Kanban is that Kanban strives to minimize the resistance by doing incremental changes whereas Scrum, figuratively speaking, compresses the process.

Moreover, Cohn (2008) noted that management is keen on adapting agile on a system level but fails to adopt agile thinking. Tanner and von Willingh (2014) listed the same deficiency as a factor causing agile projects to fail. Lilja described the exact same chain reaction using the Thinking-System-Performance illustration with Kanban. Furthermore, Strandell pointed out that Kanban led projects fall apart if they lack a shared vision and goals.

Category 3 concentrated on the differences between Kanban in reality and Kanban in theory. The differences provide fruitful insights into the potential pitfalls that Kanban faces during implementation. One of the most significant findings of the study is that implementing Kanban superficially prevents Kanban from redeeming its value proposition. The phenomenon is not new to agile circles. Brown (2012) argues the process of "going agile" must be clearly explained and outlined or otherwise the development practices will change, but in name only. What he is practically describing is the situation, which Strandell faced with the Finnish Government commission with Kanban, which turned out to be Waterfall instead. Organizations claim to use agile methods or even worse actually believe to use agile methods.

There is a similar misconception with Scrum as well. Some skeptics claim that Scrum projects fail as high as 70% of the time (Denning, 2011). Reasons for the extreme high failure rate can be explained by the "ScrumBut" phenomenon. Often a sentence starts with a "Yes, we are using Scrum but" and continues with an excuse to loosen up the rules. By cherry picking favorites from the Scrum tool bag, teams can avoid the uncomfortable but necessary pressure the process requires. Every Scrum role, rule, and time-box has a certain meaning and is designed to produce the desired outcomes and address the predictable reoccurring problems. ScrumButs mean that Scrum has exposed a dysfunction in the system that is a source of a constraint in the development chain but is too hard to fix. ScrumBut drowns the problem. Modifying Scrum to make it invisible does not eliminate the dysfunction, only hides it (Schwaber, 2011; Honkonen, 2014a). The tension Scrum creates is inevitable making it easier just to leave out the rules that cause pain and pressure within the process. The 70% tells that usually the prevailing culture of hierarchical bureaucracy is triumphant. Hence, what the

experts in this study were actually describing was “KanbanBut”. The reasons behind the superficial Kanban implementations are exactly the same. For both cases, the superficial implementation exists since facing the brutal facts is too difficult.

Category 4 includes the hidden pitfalls from theory point of view. Features those are contradictory in nature and only applicable in theory and not necessarily in action. Holler pointed out that Kanban becoming self-deliberate is a threat. Doing Kanban just for the sake of Kanban and endorsing efficiency over effectiveness. Similarly, Aastha Singh (2013) argues that following Scrum guidelines blindly will lead to the same outcome. “Scrum for Scrum”. The fact is, Scrum does not tell how to do Scrum either. One has to self-figure out what is important. There are no best practices (Mayer, 2009a).

Category 5 presented two scenarios discussing Kanban’s suitability and deficiencies creating pitfalls when applied for unsuitable situations. These pitfalls remain Kanban specific due to the uniqueness of the theory. However, agile scalability has been criticized. Interesting was that none of the informants mentioned project size as a potential pitfall of Kanban. Instead, nature of customer demand and heaviness of Kanban Method were discussed in further detail.

Note, it is trivial to count the number of the pitfalls due to the qualitative nature of the study since generalizations cannot be made based on the data. Besides the pitfalls are overlapping and linked to each other. For example, a company facing a green field situation can implement Kanban because they want to mathematically measure the project performance. The company will end up implementing Kanban on a superficial level because they are unaware of the needed agile mindset change. Since there is no warning signal the company can go on and on improving efficiency, eliminating non value-adding tasks when they are actually responding to failure demand instead of value demand.

From a broader point of view, the five categories can be described by two distinctive terms. The categories from one to three are *social* and the categories four and five are *technical*. False intentions behind the adaption of Kanban, changing the mindset and coping with Kanban in reality all relate to human behavior and misled conception of what Kanban really is. The perception of Kanban as a process management tool does not equal its real nature as a change management framework. The first three categories all share social influencers, which highlight the importance of relationships and communication. Categories four and five, on the other hand, underline the technicalities and methodology of Kanban. Both “suitability of Kanban” and “the theory of Kanban” discuss features that consider the concepts of Kanban by

definition and provide insight to situation where Kanban is not suitable due technical features such as nature of the demand.

To conclude, a common pitfall all agile methodologies struggle with, including Kanban, is the distortion of silver bullet. Schwaber (2011) has had to argue that Scrum is no silver bullet and people expecting otherwise are doomed to fail. Boehm and Turner (2004) argue that agile projects fail because people expect agile automatically to heal everything. As Strandell and Lilja emphasized, Kanban requires above all work and communication. The hardest thing is to unlearn previous habits and give Kanban time to incrementally improve the process. Cohn (2009) has recognized the same symptoms with struggling Scrum projects. Scrum does not fail – people do. The same applies for Kanban. The pitfalls are not built-in features of Kanban but the outcome of erroneous human interpretation, which is perhaps one of the most fundamental findings of this study along with inability to change mindset and superficial implementation.

7 CONCLUSIONS

The seventh and final chapter concludes the thesis by presenting a summary of the study. In addition, the chapter presents the conclusions most pertinent to the research questions. To conclude, the chapter recalls the limitations of the study and makes recommendations for further research. The final section reflects on my own experiences during the writing of the thesis.

7.1 RESEARCH SUMMARY

Agile software development has created a paradigm shift in the field of software engineering during the past two decades. Instead of fixed scope deliveries, budgets and schedules, agile methods place more value on people, interaction and working software rather than on tools, contracts and plans.

Kanban is a change management framework, which is built on lean and agile values. Kanban has been perceived as the fairy godmother of software development, which makes reasons behind struggling Kanban projects particularly interesting. The thesis studied the hidden pitfalls of Kanban by interviewing five agile experts in the software industry.

The findings were divided into five categories based on the reasons affecting the emerge of the pitfalls. The primary finding was that the inability to change mindset to support agile values prevents achieving Kanban induced benefits. By failing to change the organization's mindset a process is likely to face difficulties due change resistance. The second major finding of the study was that many software development teams claiming to be using Kanban had only made a superficial implementation of the real method. In addition, as a fundamental conclusion – the study was able to identify that the pitfalls were not inbuilt features of Kanban but the result of misled human interpretation of what Kanban really is.

7.2 CONCLUSIONS

This part of the thesis is used to provide a platform for evaluating the connotation of the results and conclusions associated with Kanban's pitfalls. I have structured the conclusion insights into five parts that have played a key role in the inception of the pitfalls.

1. KANBAN IS INSUFFICIENT BY ITSELF

Kanban is not a lifecycle methodology or a process management tool. The difference is important to understand since Kanban by itself is not a sufficient method for managing software processes. Anderson designed Kanban to be a guiding framework, which has to be integrated into an existing process tool. Mistaking Kanban for a process management tool can have devastating consequences since most of the pitfalls lead back to the false and misled comprehension of what Kanban really is. The findings endorse the view that Kanban is about creating a kaizen culture. Changing the mindset is about transforming to agile rather than adopting agile.

2. KANBAN IS NOT A SILVER BULLET

Some organizations perceive agile methodologies as the fairy godmother of software development where agile coaches can sprinkle magic dust on the troubled projects and remove years of atrophy and neglect in a single swoop of the wand. Agile benefits are touted and a project is kicked off with agile as a silver bullet. However, Kanban is on the same line with other change management methods. A recent survey conducted by the 2013 Towers Watson Change and Communication ROI found that employers say 55% of their change management initiatives meet their initial objectives. Change is always challenging regardless of context. Silver bullet is a myth.

3. KANBAN IS ABOUT PEOPLE

Even though the theory of Kanban concentrates on the value stream and process improvement, Kanban is fundamentally about people. Several if not all the pitfalls could be avoided if proper communication took place. Sharing vision and figuring out the nature of the customer demand all require extensive amount of communication.

4. KANBAN'S PITFALLS ARE NOT CONGENITAL

Perhaps one of the most comprehensive realizations of the study is the insight that the pitfalls are not built-in features of Kanban's theory. Rather, the pitfalls emerge due to inadequate human interpretation of the theory or incompetent implementation due to unwillingness to change the mindset. Agile can flourish only when the agile values are being appreciated.

5. LIMITING THE WIP IS KANBAN'S FUEL FOR IMPROVEMENT

Kanban forces teams to focus by limiting the amount of work in progress. The limitations are fuel for improvements and they should be treated as such. Limits correlate directly with

improvement. Limiting the amount of work in progress is a mechanism, which makes the development team focus on the current task, even if it is not on their turve. Instead starting to build another feature, a team has to help each other to remove blockages. Eventually the team can learn to predict similar situations and respond to them on earlier stages, even prevent them, before they fill up the work queue.

7.3 LIMITATIONS AND SUGGESTIONS FOR FURTHER RESEARCH

The study is limited to five industry experts mostly working in the Finnish software industry. Although each of the leading industry experts has a considerable amount of experience, the amount of experts still seemingly small. For further research the amount of informants could be increased in order to make generalizations.

Another limitation is the narrow introduction of other agile development methods. Due to scarce resources, the presentation of other agile methodologies was out of the scope of this study and only Scrum was introduced in order to provide perspective.

Moreover, four out of five informants worked for Reaktor. However, informants' experiences from previous employers and the fact that they work as consultants mitigate the risk of two employers being a limitation.

Even though the results of this study are not directly generalizable, they provide an insight on why Kanban driven projects might fail to redeem their value proposition and which factors effect and nourish failure. A natural continuum for this thesis would be to study how to prevent teams from stepping into the identified pitfalls. For example, having identified that Kanban lacks a warning signal - possible future research could investigate possible foolproof warning signals or ways to detect a superficial Kanban implementation immediately.

7.4 OWN REFLECTIONS

Having worked for Reaktor for almost three years now and having experienced Kanban's success first hand, my personal experience has not let much place to question the power of the method. I myself have been one of the participants of the standing ovation every time Kanban gets mentioned in a meeting.

For me, Vakkila's interview was an eye opener: he brought up the heaviness of Kanban. It had not crossed my mind, since for some reason the projects I had been involved with have shared pretty much the same characteristics. Another issue that was interesting to play around with was the difference between efficient and effective. Everything can be argued, and sometimes the line between even waste and value is blurry.

I was set out to look for pitfalls in the prioritization of the backlog, team size or WIP limitation, something concrete. I ended up finding concepts of a more fundamental nature, such as the magnitude of human interpretation. The challenges that Kanban presents lie much deeper than individual and dissonant principles of Kanban's theory. Each and every one of the pitfalls can be traced back to inability to change the way of thinking.

Finally, the nature of Kanban as an incremental improvement tool got to me, the emphasis being on the word incremental. I would like to understand how Kanban adopts to a situation, which requires drastic changes on a tight schedule. Rearranging chairs at the deck of Titanic does not do any good.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods. Review and analysis*. Espoo: VTT.

Addison, T., & Vallabh, S. (2002). Controlling software project risks: an empirical study of methods used by experienced project managers, *Proceedings of SAICSIT*, 128-140.

Ambler, S. (2002). *Agile/Lean Documentation: Strategies for Agile Software Development*. Retrieved 25 September 2014, from http://agilemodeling.com/essays/agileDocumentation.htm

Ambler, S. (2014). 2014 Agile Adoption Survey Results. Ambysoft.com. Retrieved 20 November 2014, from http://www.ambysoft.com/surveys/agileJanuary2014.html

Anderson, D. (2010). *Kanban Successful Evolutionary Change for your Technology Business.* Sequim, Washington: Blue Hole Press.

Barlow, R., & Irony, T. (1992). Foundations of statistical quality control. *Lecture Notes-Monograph Series*, 99-112.

Bar-Nahor, R. (2014). Why Scrum Projects Might Fail?. Retrieved 10 November 2014, from http://www.agilesparks.com/files/Why-Scrum-might-fail.pdf

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *Journal Of Engineering & Technology*, *2* (5).

Beck, K. (2001). *History: The Agile Manifesto*. *Agilemanifesto.org*. Retrieved 9 November 2014, from http://agilemanifesto.org/history.html

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., & Fowler, M., … Thomas, D. (2001). *The Agile Manifesto*. Retrieved 9th September 2014, from http://www.agilealliance.org/the-alliance/the-agile-manifesto/

Bell, T., & Thayer, T. (1976). Software requirements: Are they really a problem?, *Proceedings of the 2nd international conference on Software engineering (pp.61-68).* San Francisco, California.

Benington, H. (1983). Production of Large Computer Programs. *IEEE Annals Hist. Comput.*, *5* (4), 350-361. doi:10.1109/mahc.1983.10102

Bergman, B. (2008). Conceptualistic Pragmatism: A framework for Bayesian analysis?. *IIE Transactions*, *41* (1), 86-93.

Boehm, B. (1991). Software risk management: principles and practices. *IEEE Software*, *8* (1), 32-41.

Boehm B (2006) Some future trends and implications for systems and software engineering processes. Systems Engineering 9 (1): 1–19.

Boehm, B., & Turner, R. (2004). Using risk to balance agile and plan- driven methods. Computer, 36(6), 57-66. doi:10.1109/mc.2003.1204376

Brooks, F. (1982). The mythical man-month. Reading, Mass.: Addison-Wesley Pub. Co.

Brown, M. (2014). Search Results Don't adopt Agile. Blog.utest.com. Retrieved 20 November 2014, from http://blog.utest.com/?s=Don%27t+adopt+Agile+

Chapman, B., & Powell, J. (2014). Top Pitfalls of Agile Development. Retrieved 10 November 2014, from http://blog.credera.com/business-insights/top-pitfalls-agile-development/

Chhajed, D., & Lowe, T. (2008). Building Intuition. *International Series In Operations Research & Management Science*, 81-100. doi:10.1007/978-0-387-73699-0

Cleland-Huang J (2013) Are Requirements Alive and Kicking? Software. IEEE 30(3):13–15.

Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.

Cohn, M. (2008). How to Fail with Agile. Retrieved 10 November 2014, from http://www.mountaingoatsoftware.com/articles/how-to-fail-with-agile

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley.

Cohn, M. (2012). 2011 CHAOS Report Says Agile is More Successful Than Waterfall. Retrieved 12 November 2014, from http://www.mountaingoatsoftware.com/blog/agile-succeeds-three-times-more-often-than-waterfall

Cohn, M. (2014). *Scrum Product Owner: The Agile Product Owner's Role*. Retrieved 14 October 2014, from http://www.mountaingoatsoftware.com/agile/scrum/product-owner

Dedhia, N. (2005). Six sigma basics. *Total Quality Management & Business Excellence*, *16* (5), 567-574.

DeGrace, P., & Stahl, L. (1990). *Wicked problems, righteous solutions: a catalogue of modern software engineering paradigms*. Englewood Cliffs: Yourdon Press

Deming, W. (2000). *The new economics* (2nd ed.). Cambridge, Mass.: MIT Press.

Denning, S. (2011). Scrum Is A Major Management Discovery. *Forbes*. Retrieved from http://www.forbes.com/sites/stevedenning/2011/04/29/scrum-is-a-major-management-discovery/

Dubois, A., & Gadde, L. (2002). Systematic combining: An abductive approach to case research. *Journal Of Business Research*, *55* (7), 553-560.

Eriksson, P. & Kovalainen, A. (2008) *Qualitative methods in business research* (pp. 337), London: SAGE Publications Ltd.

Eskola, J., & Suoranta, J. (1991). *Johdatus laadulliseen tutkimukseen*. Rovaniemi: Lapin Yliopisto.

Favaro, J. (2002). Managing requirements for business value. *IEEE Software*, *19* (2), 15-17.

Fonecta.fi,. *Yritykset, tuotteet ja palvelut - Henkilöt - Kartat ja reitit - fonecta.fi*. Retrieved 15 September 2014, from http://www.fonecta.fi/

Fowler, M. (2005). The New Methodology. Retrieved 15 October 2014, from http://www.martinfowler.com/articles/newMethodology.html

Gloger, B. (2009). Kanban or Scrum? First Generation Agile Methods. Retrieved 18 October 2014, from http://borisgloger.com/2009/05/22/kanban-or-scrum-first-generation-agile-methods/

Goldratt, E., & Cox, J. (2004). *The goal*. Great Barrington, MA: North River Press.

Goldratt, E. (1990). *What is this thing called theory of constraints and how should it be implemented?*. Croton-on-Hudson, N.Y.: North River Press.

Gomm, R., Hammersley, M., & Foster, P. (2009). *Case study method*. 234-259. doi:10.4135/9780857024367

Gross, J., & McInnis, K. (2003). *Kanban made simple*. New York: AMACOM.

Hahn, G., Hill, W., Hoerl, R., & Zinkgraf, S. (1999). The impact of Six Sigma improvement - a glimpse into the future of statistics. *The American Statistician*, *53* (3), 208-215.

Harrington, H., & Trusko, B. (2005). Six Sigma: an aspirin for health care. *International Journal Of Healthcare Quality Assurance*, *18* (7), 487-515.

Hartman, B. (2010). Doing Agile Isnt The Same As Being Agile. Retrieved 15 September 2014, from http://www.slideshare.net/lazygolfer/doing-agile-isnt-the-same-as-being-agile

Hathaway, R. (2009). “It’s not all fun and games”. Experiences from two projects Miami. In *Lean Kanban Conference*. United Kingdom.

Hawrysh, S., & Ruprecht, J. (2000). Light Methodologies: It's Like Déjà Vu All Over Again. *Cutter IT Journal*, *13*, 3-12.

Highsmith, J., Anderson, D., Augustine, S., Avery, C., Cockburn, A., Cohn, M., …Wysocki, R. (2005). *Declaration of Interdependence*. Retrieved 10 September 2014, from http://pmdoi.org/

Highsmith, J. (2001). *History: The Agile Manifesto*. Retrieved 9 September 2014, from http://agilemanifesto.org/history.html

Hiranabe, K. (2007). Visualizing Agile Projects with Lean Kanban Boards. Retrieved from https://jazz.net/community/partners/downloads/WhitepaperVisualizingAgileProjects20080527.pdf

Hirsjärvi, S., & Hurme, H. (2008). *Tutkimushaastattelu. Teemahaastattelun teoria ja käytäntö.* Helsinki: Gaudeamus Helsinki University Press.

Hirsjärvi, S., Remes, P., & Sajavaara, P. (2009). *Tutki ja kirjoita* (13th ed.). Helsinki: Tammi.

Honkonen, S. (2014). *How to fix 90% of problems at work*. Retrieved 13 October 2014, from http://www.samihonkonen.com/how-to-fix-90-of-problems-at-work/

Honkonen, S. (Personal communication, February, 2014a)

Hopp, W., & Spearman, M. (2001). *Factory physics*. Boston: Irwin/McGraw-Hill.

Howard, D. (2003). The Basics of Statistical Process Control & Process Behaviour Charting. A User’s Guide to SPC. *Management-Newstyle*. Retrieved 28 September 2014, from http://www.flowmap.com/documents/booklets/spc.pdf

Hubbart, B. (2010). Kanban Distilled for Managers. Lean Learning. Retrieved 20 September 2014, from http://bobsleanlearning.wordpress.com/2010/07/28/kanban-distilled/

Inozu, B., Niccolai, M., Whitcomb, C., Mac Claren, B., Radovic, I., & Bourg, D. (2006). “New horizons for ship building process improvement”. *Journal Of Ship Production*, *22* (2), 87-98.

iSixSigma. (2007). *Six Sigma Saves Fortune 500 $427 Billion*. iSixSigma Magazine.

Jonasson, H. (2008). *Determining project requirements*. Boca Raton: Auerbach Publications.

Jones, C. (1996). The economics of software process improvement. *Computer*, *29* (1), 95-97. doi:10.1109/2.481498

Kee, W. (2006). Future Implementation and Integration of Agile Methods in Software Development and Testing. *Innovations In Information Technology*. doi:10.1109/innovations.2006.301945

Kniperg, H. (2014). *Kanban vs Scrum - How to make the best of both* (1st ed.). Crisp. Retrieved from http://www.crisp.se/henrik.kniberg/Kanban-vs-Scrum.pdf

Koskinen, I., Alasuutari, P., & Peltonen, T. (2005). *Laadulliset menetelmät kauppatieteissä*. Tampere: Vastapaino.

Ladas, C. (2008). *Scrumban*. Seatle, WA: Modus Cooperandi Press.

Lapham, M. (2012). DoD Agile Adoption: Necessary Considerations, Concerns, and Changes. Retrieved 4 October from http://www.crosstalkonline.org/storage/issue-archives/2012/201201/201201-Lapham.pdf

Larman, C., & Vodde, B. (2009). Lean Primer. *1*, 1-46. Retrieved 5 October 2014 from http://www.leanprimer.com/downloads/lean_primer.pdf

Larman, C. (2004). *Agile and iterative development* (p. 27). Boston, Mass: Addison-Wesley.

Larman, C., & Basili, V. (2003). Iterative and incremental development: A brief history. *Computer, 36* (6), 47-56.

Latzko, W. (2005). *Notes on the Six Sigma Concept by William J. Latzko*. Retrieved 8 October 2014, from http://www.latzko-associates.com//Publications/SIX_Sig.pdf

Lee-Mortimer, A. (2006). Six Sigma: a vital improvement approach when applied to the right problems, in the right environment. *Assembly Automation*, *26* (1), 10-17.

Leffingwell, D., & Reinertsen, D. (2011). *Agile software requirements*. Upper Saddle River: Addison-Wesley.

Leffingwell, D. (2007). *Scaling software agility*. Upper Saddle River, NJ: Addison-Wesley.

Liker, J. (2004). *The Toyota way*. New York: McGraw-Hill.

Marx, M. (2007) Why Should a Company Implement Six Sigma?. Retrieved 21 October 2014, from http://www.sixsigma.com/why-six-sigma/

Mayer, T. (2009). Scrum and Kanban - different animals. *Agile Anarchy*. Retrieved 25 October from http://agileanarchy.wordpress.com/2009/09/22/scrum-kanban/

Mayer, T. (2009). Scrum: A New Way of Thinking. Agile Anarchy. Retrieved 10 November 2014, from https://agileanarchy.wordpress.com/scrum-a-new-way-of-thinking/

McCauley, R. (2001). Agile development methods poised to upset status quo. *ACM SIGCSE Bulletin*, *33* (4), 14. doi:10.1145/572139.572150

McNabb, D. (2002). *Research methods in public administration and nonprofit management*. Armonk, N.Y.: M.E. Sharpe.

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

Middleton, P., & Sutton, J. (2005). *Lean software strategies*. New York.: Productivity Press.

Middleton, P. (1993). Just-In-Time Software Development. *In Achieving Software Quality in Software* (pp. 49-56). Venice, Italy: Consorzio Quality I.E.I.-CNR.

Middleton, P., & Joyce, D. (2012). Lean software management: BBC Worldwide case study. Engineering Management, *IEEE Transactions*, 59 (1), 20-32.

Miles, M., & Huberman, A. (1994). *Qualitative data analysis an expanded sourcebook* (p. 352). Thousand Oaks: SAGE.

Miller, G. (2001). The Characteristics of Agile Software Processes. In *The 39th International Conference of Object-Oriented Languages and Systems*. Santa Barbara, CA.

Oakland, J., & Followell, R. (1990). *Statistical process control*. Chapter 2. Oxford: Butterworth-Heinemann.

Ohno, T. (1988). *Toyota production system*. Cambridge, Mass.: Productivity Press.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Boston, Massachusetts: Addison Wesley.

Reaktor. (2014). *The Design of Business, Services & Technology*. Retrieved 9 September 2014, from http://reaktor.fi/

Reinertsen, D. (2009). *The principles of product development flow*. Redondo Beach, California: Celeritas.

Rook, S. (2010). Kanban: Definition of Lead Time and Cycle Time. Retrieved 3 October 2014, from http://stefanroock.wordpress.com/2010/03/02/kanban-definition-of-lead-time-and-cycle-time/

Royce, W. (1970). Managing the development of large software systems, *26* (8).

Sahota, M. (2011). *How to Make Your Culture Work with Agile, Kanban & Software Craftsmanship*. Retrieved 15 September 2014, from http://agilitrix.com/index.php?s=How+to+Make+Your+Culture+Work+with+Agile%2C+Kanban+%26+Software+Craftsmanship

Schonberger, R. (1982). *Japanese manufacturing techniques*. New York: Free Press.

Schragenheim, E., & Dettmer, H. (2000). Simplified Drum-Buffer-Rope A Whole System Approach to High Velocity Manufacturing. *Goal Systems International*. Retrieved from http://www.goalsys.com/books/documents/S-DBRPaper.pdf

Schwaber, K. (2004). *Agile project management with Scrum*. Redmond, Washington: Microsoft Press.

Schwaber, K. (2011). Scrum Fails? Telling It Like It Is. Retrieved 10 November 2014, from http://kenschwaber.wordpress.com/2011/04/07/scrum-fails/

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall.

Schwaber, K., & Sutherland, J. (2012). *Software in 30 days*. Hoboken, N.J.: John Wiley & Sons.

Schwaber, K., & Sutherland, J. (2013). *Scrum Guide*. Retrieved 14 September 2014, from http://scrumguides.org/scrum-guide.html

State of Scrum, (2014). Retrieved 12 November 2014, from https://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/Files%20and%20PDFs/State%20of%20Scrum/2013-State-of-Scrum-Report_062713_final.pdf

Shalloway, A., Beaver, G., & Trott, J. (2010). *Lean-agile software development*. Upper Saddle River, NJ: Addison-Wesley.

Shingo, S., & Dillon, A. (1989). *A study of the Toyota production system from an industrial engineering viewpoint*. Cambridge, Mass.: Productivity Press.

Singh, A. (2013). Why Do Companies Fail in Adopting Agile Practices? Retrieved 10 November 2014, from https://www.scrumalliance.org/community/articles/2013/december/why-companies-fail-in-adopting-agile-practices

Singh, J. (2013). What is agile software development model. Retrieved 18 October 2014, from http://www.jagjot.com/2013/01/agile-software-development/

Slaughter, S., Harter, D., & Krishnan, M. (1998). Evaluating the cost of software quality. Commun. ACM, 41(8), 67-73. doi:10.1145/280324.280335

Sprott, D. (2014). Agile and the Fairy Godmother. Retrieved 13 October 2014, from http://davidsprottsblog.blogspot.fi/2014/07/agile-and-fairy-godmother.html

Standish Group. (1994). Retrieved 12 November 2014, from Standish. The CHAOS Report. http://www1.standishgroup.com/sample_research/chaos_1994_1.php

Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review, 64* (1), 137-146.

Tanner, M., & von Willingh, U. (2014). Factors leading to the succ ess and failure of agile projects implemented in traditionally waterfall environments. In Human Capital without Borders: Knowledge and Learning for the Quality of Life. Portoroz, Slovenia: Make Learn.

Thomas, D. (2010). *Agile 2010 Keynote by Dave Thomas - Catalyst - Agile & Culture*. Retrieved 18 September 2014, from http://agilitrix.com/2010/08/agile-2010-keynote-by-dave-thomas/

Towers Watson,. (2013). *Quarter of Employees Gain from Change Management Intiatives*. Retrieved 20 October 2014, from http://www.towerswatson.com/en/Press/2013/08/Only-One-Quarter-of-Employers-Are-Sustaining-Gains-From-Change-Management

Toyota-global.com. (2014). *Just-in-Time*. Retrieved 25 October 2014, from http://www.toyotaglobal.com/company/vision_philosophy/toyota_production_system/just-in-time.html

Trapani, K. (2014). Adopting Agile | cPrime. Retrieved 20 November 2014, from https://www.cprime.com/tag/adopting-agile/

Tuomi, J., & Sarajärvi, A. (2006). *Laadullinen tutkimus ja sisällönanalyysi*. Jyväskylä: Tammi.

Urbanski, J. (2013). Find out the benefits of Agile Methodology from a Scrum Master. *eDigital*. Retrieved from http://edigitalaus.blogspot.fi/2013/08/agile-methodology-q-with-justin.html

Uusitalo, H. (1991). *Tiede, tutkimus ja tutkielma*. Porvoo: WSOY.

Walshe, K., Harvey, G., & Jas, P. (2010). *Connecting knowledge and performance in public services*. Cambridge: Cambridge University Press.

West D., Grant T., Gerush, M. & D'Silva, D. (2010). Agile development: Mainstream adoption has changed agility. Forrester Research.

Willeke, E., Anderson, D., & Landes, E. (2009). Proceedings of Lean & Kanban Software Conference. In *Lean & Kanban Software Conference*. Bloomington: Wordclay.

Womack, J., Jones, D., & Roos, D. (1990). *The machine that changed the world*. New York: Rawson Associates.

Womack, J., & Jones, D. (1997). *Lean thinking*. New York, NY: Simon & Schuster.