View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Helsingin yliopiston digitaalinen arkisto

Department of Computer Science
Series of Publications A
Report A-2011-4

Lean Thinking in Software Development: Impacts of Kanban on Projects

Marko Ikonen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, University Main Building, on 19th December 2011, at noon.

University of Helsinki
Finland

Supervisors

Professor Pekka Abrahamsson (University of Helsinki, Finland)
Professor Jukka Paakki (University of Helsinki, Finland)

Pre-examiners

Professor Giancarlo Succi (Free University of Bolzano-Bozen, Italy)
Professor Juan Garbajosa (Technical University of Madrid, Spain)

Opponent

Professor Markku Oivo (University of Oulu, Finland)

Custos

Professor Pekka Abrahamsson (University of Helsinki, Finland)

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.helsinki.fi
URL: http://www.cs.Helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

Copyright © 2011 Marko Ikonen
ISSN 1238-8645
ISBN 978-952-10-7409-7 (paperback)
ISBN 978-952-10-7410-3 (PDF)
Computing Reviews (1998) Classification: D.2.9, K.6.3
Helsinki 2011
Unigrafia

Lean Thinking in Software Development: Impacts of Kanban on Projects

Marko Ikonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
marko.ikonen@cs.helsinki.fi
http://www.cs.helsinki.fi/u/mjikonen/

PhD Thesis, Series of Publications A, Report A-2011-4
Helsinki, December 2011, 104+90 pages
ISSN 1238-8645
ISBN 978-952-10-7409-7 (paperback)
ISBN 978-952-10-7410-3 (PDF)

Abstract

The history of software development in a somewhat systematical way has been performed for half a century. Despite this time period, serious failures in software development projects still occur. The pertinent mission of software project management is to continuously achieve more and more successful projects. The application of agile software methods and more recently the integration of Lean practices contribute to this trend of continuous improvement in the software industry. One such area warranting proper empirical evidence is the operational efficiency of projects. In the field of software development, Kanban as a process management method has gained momentum recently, mostly due to its linkages to Lean thinking. However, only a few empirical studies investigate the impacts of Kanban on projects in that particular area. The aim of this doctoral thesis is to improve the understanding of how Kanban impacts on software projects. The research is carried out in the area of Lean thinking, which contains a variety of concepts including Kanban. This article-type thesis conducts a set of case studies expanded with the research strategy of quasi-controlled experiment. The data-gathering techniques of interviews, questionnaires, and different types of observations are used to study the case projects, and thereby to understand the impacts of Kanban on software development projects. The research papers of the thesis are refereed, international journal and conference publications. The results highlight new findings regarding the application of Kanban in the software context. The key findings

of the thesis suggest that Kanban is applicable to software development. Despite its several benefits reported in this thesis, the empirical evidence implies that Kanban is not all-encompassing but requires additional practices to keep development projects performing appropriately. Implications for research are given, as well. In addition to these findings, the thesis contributes in the area of plan-driven software development by suggesting implications both for research and practitioners. As a conclusion, Kanban can benefit software development projects but additional practices would increase its potential for the projects.

Computing Reviews (1998) Categories and Subject Descriptors:

D.2.9 Software Engineering: Management—*productivity, programming teams, software process models*

K.6.3 Management of Computing and Information Systems: Software Management—*software development, software process*

General Terms:

Human Factors, Management

Additional Key Words and Phrases:

Kanban, Lean thinking, Lean software development, project success, software development project

Acknowledgments

Having proceeded to the end of an amazing journey in my one life-learning experience, the time has come to take a breath and to look at what I have done. As always, this kind of journey requires sacrifices in the name of science. Long days, even longer nights, huge amounts of studying and lots of work in order to find new and interesting discoveries on an international scale. Meanwhile, those days are taken away from close friends and from your own leisure. Some critical decisions have had to be made regarding which things should be prioritized over others. After these days, nights, and decisions, here it is, namely, my doctoral dissertation. Even though I have felt lonely sometimes during the process, creating a dissertation is teamwork. When I look at my dream team, I truly can state that I have had great luck in enjoying all of the members' contribution, friendliness, and helpfulness toward me and my "baby". The time has come to end this wonderful journey, to proceed forward, and, the most pleasant part, to utter commendations to my dear colleagues and friends who participated in this delightful journey.

First of all, I would like to express the sincerest gratitude to my dear supervisors: The invaluable contribution of Professor Jukka Paakki enabled me to start the thesis process, and when Professor Pekka Abrahamsson joined the team later I could not have wished for a better pair of supervisors for my scientific firstborn, my doctoral thesis. Both professors are shining examples for the university of how deeply committed guidance can give wonderful results. I will never stop being amazed at how, every time I had questions about my thesis work or a seemingly impossible problem, my supervisors always found time for me – whether they were heading a department or a faculty, or otherwise had their hands full of work. The result of all this generosity is that, after careful preparation, all my internationally refereed articles that are incorporated into this thesis were finished within 18 months. Without my two strong supervisors this would hardly have been possible. These warm and dedicated professors are an example of what scientific cooperation can be at its best.

Former 1st Vice-Rector of the university, Professor Hannele Niemi, did not hesitate to lavish her time on me, either; she gained an important role for me when I worked in the IQ Form project, headed by her, in the final stages of my Master degree programme. It is mostly thanks to her that I decided to continue with my postgraduate degree in the first place.

Head of Studies at the Department of Computer Science, Jaakko Kurhila, has also held an important role both because of his support for my postgraduate studies and for enabling me to start publishing my scientific work and supporting me throughout my thesis work.

Eteläsuomalainen osakunta (ESO, student union for southern Finland) is another organization worth mentioning, as I have had the privilege of spending time with academic friends from all disciplines from my freshman year to this day. The Inspector Emeritus of the union, current Vice-Rector of the university, Professor Kimmo Kontula and his lovely wife, Inspectrix Emerita Heljä Kontula have supported my academic career.

Furthermore, the valued pre-examiners of my thesis, Professor Giancarlo Succi (Free University of Bolzano-Bozen, Italy) and Professor Juan Garbajosa (Technical University of Madrid, Spain) deserve my heartfelt gratitude. Because of the printing date of the thesis, I will thank my esteemed opponent, Professor Markku Oivo (University of Oulu) in advance.

The co-authors of the research papers who were not mentioned above, i.e., Petri Kettunen, D.Sc. (Tech), Nilay Oza, PhD, Fabian Fagerholm, Henri Karhatsu, and Elena Pirinen deserve my compliments. In spite of all the hardship, causing occasional feelings of loneliness, which I have experienced during my thesis work, I have never been alone, but surrounded by the academic community, my dear colleagues and friends. Since it is impossible to mention everyone by name or enumerate the ways in which they have supported me and influenced my thesis, I want to offer my thanks to everyone here and now, and I hope these thanks will find their marks without further ado.

Naturally, my parents Risto and Raili play the most significant role in the finishing of my thesis, as it would have been virtually impossible for me to embark on this academic adventure without them.

This thesis work has been funded by the Graduate School on Software Systems and Engineering (SoSE) in 2007–2009, the Cloud Software program as a part of TIVIT (funded by TEKES) in 2010, and the Scalable High-performing Software Design Teams (SCABO) project (funded by TEKES) in 2011.

Helsinki, November 26th 2011
Marko Ikonen

List of Original Publications

This doctoral thesis is based on the following six original research papers that are refereed international journal and conference publications.

Research Paper I: Ikonen, M.[1] and Kurhila, J. (2009). Discovering high-impact success factors in capstone software projects. In *Proceedings of the 10th ACM conference on SIG-information technology education*, SIGITE '09, pages 235–244, New York, NY, USA. ACM.
Cited in this Thesis as Paper I.

Research Paper II: Ikonen, M.[2] and Abrahamsson, P. (2011). Operationalizing the concept of success in software engineering projects. *International Journal of Innovation in the Digital Economy (IJIDE)*, 2(3):11–37. IGI Global.
Cited in this Thesis as Paper II.

Research Paper III: Karhatsu, H., Ikonen, M.[3], Kettunen, P., Fagerholm, F., and Abrahamsson, P. (2010). Building blocks for self-organizing software development teams: A framework model and empirical pilot study. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE '10)*, volume 1, pages V1-297–V1-304. IEEE.
Cited in this Thesis as Paper III.

[1] Ikonen conducted the study and measured all the data and made the interviews and analyzes. The paper was written and ideated mainly by him with the valuable suggestions of the coauthor. The coauthor's deep knowledge in Computer Science Education was utilized.

[2] The paper was written mostly by Ikonen while both authors contributed equally to the composing of the theme. Ikonen conducted the study and measured all the data and made the interviews. The suggestions and contribution to the analysis from the coauthor were valuable.

[3] Ikonen contributed significantly to the design and completeness of the paper. He also refined the framework and was equally involved in the writing and finalizing process of the paper with the coauthors.

Research Paper IV: Ikonen, M.[4], Kettunen, P., Oza, N., and Abrahamsson, P. (2010). Exploring the sources of waste in Kanban software development projects. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA '10)*, pages 376–381, Los Alamitos, CA, USA. IEEE Computer Society.
Cited in this Thesis as Paper IV.

Research Paper V: Ikonen, M.[5], Pirinen, E., Fagerholm, F., Kettunen, P., and Abrahamsson, P. (2011). On the impact of Kanban on software project work: An empirical case study investigation. In *Proceedings of the 16th International Conference on Engineering of Complex Computer Systems (ICECCS '11)*, pages 305–314, Los Alamitos, CA, USA. IEEE Computer Society.
Cited in this Thesis as Paper V.

Research Paper VI: Ikonen, M. (2010). Leadership in Kanban software development projects: A quasi-controlled experiment. In Abrahamsson, P. and Oza, N., editors, *Proceedings of the 1st International Conference on Lean Enterprise Software and Systems (LESS 2010)*, volume 65, Part 2 of *Lecture Notes in Business Information Processing*, pages 85–98, Berlin Heidelberg. Springer-Verlag.
Cited in this Thesis as Paper VI.

[4]Ikonen conducted the study and made all the interviews. The writing process and analyzing was contributed equally by Ikonen and the second author while everyone participated in the refining and finalizing of the paper, coordinated by Ikonen.

[5]Ikonen significantly contributed to the theme, structure, and finalizing of the paper. The authors together refined the framework and wrote the paper. The data were gathered and interviewed by Ikonen and the second author.

Contents

Chapter 1

Introduction

During the mid-nineties, initial requirements and documentation steps were found to be frustrating and difficult to implement in practice in software development projects. Meanwhile, plans and requirements of the projects were reported to expire in a short period of time (Williams and Cockburn, 2003).

Systematic engineering approaches, including plan-driven software development methods, were no longer considered to be the best way to produce software due to their problems (Takeuchi and Nonaka, 1986). Agile software development practices (Abrahamsson et al., 2002) were challenging this conventional philosophy in the topic. Regardless, agile methods, such as Scrum (Sutherland, 1995)[1], have been reported to have their own problems. Requirements traceability, for example, is not able to be performed in an agile approach as it is established in conventional requirements management (such as ISO/IEC 12207:2008) (Espinoza and Garbajosa, 2011). Moreover, years after executives who have attempted to solve problems with the aid of the agile approach, still encounter the same complaints and desires, as Anderson (2010) states: (1) their technology teams still suffer with unreliableness, (2) their business is still too unresponsive and has not acquired the agility anticipated, and (3) costs are still out of control. On the other hand, Díaz et al. (2009) show how the CMMI oriented organizations can map their practices to Scrum in order to improve their software processes. Instead of such combining, *Lean thinking* has been adapted into

[1]The first references in the literature related to the term *Scrum* point to the article of Takeuchi and Nonaka (1986) which presents an adaptive, quick, self-organizing product development process originating from Japan (Schwaber and Beedle, 2002). Schwaber (1995) then presents "Scrum Development Process" in Section 4.10 of the OOPSLA '95 Workshop on Business Object Design and Implementation. The author of the workshop is Sutherland (1995).

software engineering (Shalloway et al., 2009, 78–94). While the successful history of Lean thinking, which gives the motivation for applying it to software development, originates in the 1950s from Toyota's car manufacturing, it is a relatively novel philosophy in software engineering (Poppendieck and Poppendieck, 2003). *Kanban* executes the Lean thinking in practice (Becker and Szczerbicka, 1998; Chai, 2008). It is one of the key operation management tools in Lean manufacturing (Liker, 2004, 176). It drives project teams to visualize the workflow, limit work in progress (WIP) at each workflow stage, and measure the cycle time (i.e., average time to complete one task) (Kniberg, 2009). There is a strong practitioner-driven movement supporting the idea of the use of Kanban in software engineering (Hiranabe, 2008; Shinkle, 2009; Shalloway et al., 2009). It is evident that the outcomes of applying Kanban are expected to be high in software development, as they have been in manufacturing. This expectation is reasonable because of Kanban's adaptability (it welcomes changes on the requirement list), visualization (it eases management by visualizing the progress), and prerequisites for its successful use (it drives team members to cooperation and communication, among others) (Kniberg, 2009). This is not yet, however, confirmed by means of empirical studies: only a few studies have empirically explored the dynamics of Kanban from the viewpoint of software development. In addition, the relation between Kanban and *waste*[2] in software development is, according to our knowledge, an area of study that has not received much attention yet.

The software industry is in constant search of new ways of solving existing problems (Jacobson, 2009). The goals vary in different improvement initiatives ranging from resolving time-to-market delay to reducing operation costs and increasing productivity. The conventional way of developing software has widely been criticized (Larman and Basili, 2003; Poppendieck and Poppendieck, 2003; Ramasubbu and Balan, 2009). This criticism questions the conventional belief in the ability to anticipate all the problems at the beginning of the project: the outputs of this anticipation are schedules and requirements, likely unrealistic ones because the customer and the producer may not have guessed all the necessary requirements of the customer at the beginning of the project. Shalloway et al. (2009, 1), for example, argue that unlike many other disciplines, software development usually does not start with clearly defined requirements. Rather than a clearly defined, up-front plan, reaching the goal is more a process of discovery (Royce, 1970;

[2]Waste can be understood as an unnecessary action that does not add value for the customer or another representative that the work is done for. For more details, see Chapter 2.2.1.

Espinoza and Garbajosa, 2011; Ramasubbu and Balan, 2009).

After all, a difference between theory and practice that makes the field even more complex is the uncertainty. De Meyer et al. (2002) refer to four types of uncertainty: variation, foreseen uncertainty, unforeseen uncertainty, and chaos. Uncertainty is a rule, not an exception, and the greatest chance to produce a successful project lies in companies which understand this rule (De Meyer et al., 2002). Moreover, managing a software project is complex because it is more than a formally designated set of tasks (Jurison, 1999). Typical software-related problems concerning management are as follows: (1) Intangibility: software is difficult to manage without visible milestones to measure quality and progress. (2) Complexity: the sheer complexity of software is not so understood and this leads not only to technical but management problems, too. (3) The volatility of requirements: the pressure for software changes is high because software can be changed more easily than hardware (Jurison, 1999).

Some project failures can be traced to malfunctioning group performance, which is caused by inadequate attention to people and teamwork issues (Jurison, 1999). People have a tendency to concentrate on the technical aspects (hardware and software) rather than the peopleware and impermanence (Gunson et al., 2003). In software engineering generally, the correlation exists between contribution and social motivation without interfering economics (Bonaccorsi and Rossi, 2004; Jensen and Scacchi, 2005). Watson (2006, 12) states that "Managerial activities are always and inevitably implicated in issues of power and relative advantage and disadvantage between human groups and human individuals."

Understanding is required on how human and organizational factors affect the execution of software development tasks (Weinberg, 1971; Scacchi, 1984; DeMarco and Lister, 1987). The laws of physics are invalid for human-oriented software processes (Pedrycz and Succi, 2007). In addition to any area-specific skills and general management experience, the following three characteristics are essential for a project manager: knowledge (about project management), performance (ability to work while applying knowledge of project management), and personality (leadership, attitudes, and core personality in personal behavior when performing the project) (PMI, 2008, 13). Operating in a modern customer-driven business environment provides a balanced mixture of soft and hard skills (Yasin et al., 2002).

The reasons mentioned above indicate that focusing on process issues without notifying people is inadequate for the success of software projects. This suggests focusing on people-oriented processes and hereby leads to set the research questions of this thesis. In addition to establishing the

research questions, the following sections set the boundaries for the scope of the research, and describe the structure of the thesis.

1.1 Research Questions

Examples of the challenges mentioned above indicate that a conventional plan-driven approach is not the solution. Therefore, this doctoral thesis explores other ways to perform software development.

One of the newest fashions in the software industry is the attempt to apply Lean production principles to software development. One key trait of these principles is eliminating all kinds of waste from the development. Moreover, respecting people and empowering teams are fundamental in Lean thinking and they relate to the concept of self-organizing teams. Consequently, similar ideas for waste removal and people respect have been proposed to be adopted for software product development (Poppendieck and Poppendieck, 2003).

Kanban has been applied to software production as a project management process model (Hiranabe, 2008). The buzz is already in motion on combining Kanban with the well established Scrum method in agile software development. Kniberg (2009), for example, introduces similarities and differences between Kanban and Scrum while Ladas (2009) combines parts of Kanban and Scrum in practice. The low number of empirical studies on exploring the internal dynamics and the process impact of Kanban in software engineering, however, leads us to the research question that is set as follows.

RQ1: How does Kanban impact on software development projects?

In order to answer this question, this thesis divides it into three sub-questions. First, an understanding is needed on what prevents or supports projects going forward. Second, since the literature suggests focusing not only on processes but people too, self-organization has to be explored for comparability of the results between the case projects. This self-organization is preferred by Kanban that supports Lean manufacturing. Third, based on these two sub questions, we need to know how Kanban influences the progress of projects from the viewpoints of waste, project work, and management. Focusing on Lean-based Kanban is reasonable since the

literature addressed above suggests that practices being used in software development still have drawbacks. The three research sub-questions are set as follows.

RQ1.1 Which constructs of software development projects affect the outputs?

In order to study Kanban's impact on software development projects, we need a framework for issues affecting project outputs. The literature of software engineering projects has focused on identifying isolated success factors even though the issues behind the success rely on multidimensional constructions with multidisciplinary factors (Ikonen and Abrahamsson, 2010). While the term of project success is understood in a variety of ways (Agarwal and Rathod, 2006), some definitions taking the customer into account can be agreed to have an established position. Examples of such definitions are the four dimensions of project success of Shenhar et al. (2001) and a definition for success by the Project Management Institute in its PMBOK[3] guide.

While the concept of success is often overlooked, the concept of project management is equally little addressed theoretically. Koskela and Howell (2002) argue that there is no explicit theory for project management but rather that the theoretical foundation for project management has emerged from the works of the Project Management Institute (see PMI (2008)). As a resolution, Koskela and Howell (2002) propose a novel theory of project, which portrays a project by a transformation view on operations. They suggest that in the transformation view, a project is conceptualized as a transformation of inputs to outputs. A project, then, is managed with a number of principles.

As one of the guiding principles, Koskela and Howell (2002) identify the principle of decomposing, which aims at decomposing the total transformation hierarchically into smaller transformations, tasks, and minimizing the cost of each task independently. From the software viewpoint, this is not a novel realization. In the most basic software engineering textbooks, such as Sommerville (2007), a software project is viewed conventionally as a problem of decomposition. The challenge that the software project brings into the project concept is the significant role of the human in the process of software development (e.g., Boehm (1981); Clegg et al. (1996)). Besides, this kind of development is a group activity (Tolvanen, 1998, 42) wherein multiple people participate in different roles. This means that, in addition to individual points of view, the focus must include the team level.

[3] A Guide to the Project Management Body of Knowledge (PMBOK) (PMI, 2008).

Moreover, transformation processes from conventional to Lean in software engineering are not tangible artifacts as in other engineering fields. Cockburn (2002) suggests that what flows in software development is a series of invalidated decisions that become validated only after they have been implemented. Thereby, it can be suggested that any model aiming at depicting a software project success model should take these characteristics into account.

Finally, as a solution for problems related to software development and recognized in the literature, Lean principles are suggested to apply to the area (Poppendieck and Poppendieck, 2003). These principles, in contrast to conventional software development, take into account critical issues, such as seeing the whole, empowering the teams, and eliminating waste. Hence, also the human aspect appears to have been concerned in Lean thinking.

Having concerned the variety of isolated success factors, the problem of the research question can be approached by first determining factors that drive or restrain the progress of projects. Then, by categorizing these factors as constructs, a holistic picture of their effects can be formulated.

RQ1.2 What are the key elements of self-organizing teams?

Self-organization has emerged as an important area of study also within software engineering management, in particular with agile methods (Abrahamsson et al., 2002; Moe et al., 2009a,b). In addition, self-organizing is a fundamental part of Lean thinking as mentioned above (i.e., the Lean principle of empowering the teams, for example). For this reason, the research question focuses on self-organizing teams instead of non-self-organizing teams. Self-organizing teams benefit organizations (Behnke et al., 1993; Guzzo and Dickson, 1996; Janz, 1998). Positive outcomes are often related to performance effectiveness, member attitudes, and behavior (Cohen and Bailey, 1997). Self-organizing teams can react to problems quickly since the decision-making is close to the problem (Tata and Prasad, 2004). Instead of waiting for a manager's approval, such a team has the authority to take necessary actions by itself (Moe et al., 2009a).

Self-organization has emerged as an important area of study also within software engineering management, in particular with agile methods such as Scrum. Despite the often claimed benefits, a shortage of conclusive empirical studies exists in the area. Some results indicate no connection between empowerment and success or that the project performance is not increased (Reilly and Lynn, 2003). This contradiction indicates that self-organization is not a panacea. Just calling a group self-organizing does not automatically translate into better performance. The organizational

context like the reward system, supportive leadership, training, available resources, and the structure of the organization influence how teams can self-organize and perform (Cohen and Bailey, 1997; Tata and Prasad, 2004).

The problem of the research question can be approached by first conducting an experimental, literature-based research model and then by evaluating this model empirically. The model can then be used for better understanding of what makes a self-organizing team successful and how to build such a team.

RQ1.3 What are the salient characteristics of Kanban that affect the progress of software development projects?

The Kanban process model (Gross and McInnis, 2003) executes Lean thinking in practice (Becker and Szczerbicka, 1998; Chai, 2008) and is one of the key operation management tools in Lean manufacturing (Liker, 2004, 176). Moreover, it provides a way to prevent waste, which is considered the most important Lean principle (Poppendieck and Poppendieck, 2003). This waste, such as unreasonableness or inventories, does not add value for the customer. Thus, eliminating waste from projects is reasonable. Questions, however, remain.

First, waste in software development, nevertheless, is mostly invisible when compared with a pipeline in manufacturing (Poppendieck and Poppendieck, 2003). Second, Kanban's relation to waste is an area of study that has not received much attention in software development. In other words, it is not known whether Kanban benefits software development projects wherein waste is more likely abstract than physical. Third, Kanban does not seem to intervene in management despite its importance. Regarding the self-organizing team principle, a relevant question in order to improve project performance is, whether management is still necessary or it is only a waste of time and resources. Kotter (1996, 7) and Watson (2002, 276–319), however, claim that without leadership or management even a capable staff head cannot reach goals well enough. The lack of a clear authority structure in software development is both a cause of chaos and freedom (Jensen and Scacchi, 2005).

Thereby, investigating impacts of Kanban on software project work is needed for answering the research question. The problem of the research question can be approached by first conducting research models wherein waste[4] and work aspects (such as documentation) are salient points. Then, by evaluating these models, influences of Kanban on the progress of the projects can be illustrated.

[4]The term is introduced in Chapter 2.2.1.

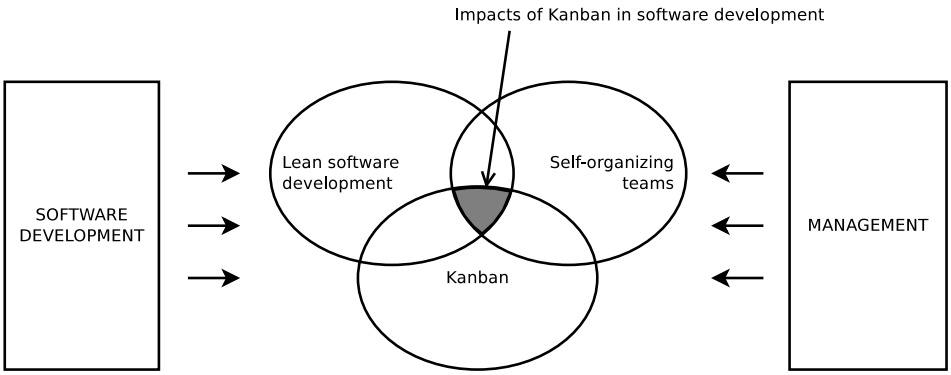

Figure 1.1: The scope of this thesis.

1.2 Scope of the Research

This doctoral thesis combines the research areas of software development and software project management. A reason for this combining is the fact that issues behind project success depend on multidimensional constructions with multidisciplinary factors, as mentioned above. This thesis calls into question some conventional practices in software development. According to Clegg et al. (1996), for example, software development can be characterized as a series of sequentially organized phases of activities, such as design, programming, and maintenance. While it can be characterized so, controlling long phases against time in the rapidly changing area is risky (Williams and Cockburn, 2003). The rapid adaptability into continuously changing circumstances is one characteristics of Lean thinking that is challenging the conventional philosophy (Poppendieck and Poppendieck, 2003).

Figure 1.1 illustrates the scope of the research which is the gray intersection of the three areas. The areas described below work as boundaries for this scope. The concept of *software development* in this thesis relies on the definition of Humphrey (1995, 4–5) regarding the software process:

> "The software process is the sequence of steps required to develop or maintain software... More specifically, the software process sets out the technical and management framework for applying methods, tools, and people to the software task."

Management, instead, refers to the definition of Kotter (1996, 25–26):

> "Management is a set of processes that can keep a complicated

> system of people and technology running smoothly. The most important aspects of management include planning, budgeting, organizing, staffing, controlling, and problem solving."

Good management cannot guarantee project success but bad management often results in project failure, such as delayed software delivery, exceeded budgets, and unmet requirements (Sommerville, 2007, 93). Traditionally, the role of a project manager has included activities such as planning, delegating, resourcing, monitoring and reporting (Pressman, 1997, 24–25, 59–74). Blanchard (2001), however, argues that the leader's role has shifted dramatically: in the past, the leader was emphasized as boss but today, leaders can no longer lead with position power alone. Rather, they have to be partners with their team. The "command-and-control" role of judging and evaluating has to be switched off (Blanchard, 2001).

Lean software development, based on Lean thinking (Ohno, 1988; Shingo, 1989; Womack and Jones, 2003) (Chapter 2), is one of the three areas of interest in the scope of this thesis. The second area is the Kanban (Chapter 2.5) way of doing things, which is also a foundational part of Lean thinking. The third area is self-organization (Chapter 5.3), which also relates to Lean thinking. Lean thinking, in contrast to agile methods, has not been applied to the area of software development until recently.

A self-organizing team in this thesis refers to the definition for an autonomous work group by Guzzo and Dickson (1996):

> "Autonomous work groups are teams of employees who typically perform highly related or interdependent jobs, who are identified and identifiable as a social unit in an organization, and who are given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions with economic consequences (usually up to a specific limited value)."

These three areas combine software development and management into a comprehensive base that is studied in this thesis.

1.3 Thesis Structure

This doctoral thesis is based on the following six original research papers (cited as Paper I, II, III, IV, V, and VI) that are refereed international journal and conference publications. In order to explore the research phenomenon outlined in this thesis, each paper contributes to the increased understanding of improvement in software development. By following this

theme, the structure of this thesis has been modeled upon the research papers.

Paper I creates the motivation for the research by showing that certain factors of project success exist before the end part of projects. Due to the fragmented area of project success in software engineering, Paper II explores the area and suggests a framework for project success. After having evaluated this framework, further studies are conducted in order to examine the Kanban-based Lean software development with the aid of the framework. Due to this focus, the case projects consist of self-organized software development teams. Thereby, Paper III explores self-organization. Then, Paper IV, V, and VI explore the impacts of Kanban on software development projects regarding waste, project work, and management. Together, applying the results of Paper I, II, and III to Paper IV, V, and VI provides new information regarding how the Kanban-way of doing things affects software development.

The summary part of the thesis is structured as follows. This Chapter 1 describes the research questions, scope, and structure of the thesis. Chapter 2 reviews the background of Lean thinking from its origin to applying it to software development. Lean concepts are explained as well. Moreover, the Kanban method is introduced and linked into the Lean context. Chapter 3 is two-fold. First, it considers project success. Second, it presents a framework for project success. In addition, this framework is mapped with the Lean software development. Chapter 4 sets out the empirical research design including the research approach, methods, and context in which the research took place. Chapter 5 briefly states the research contribution of the empirical research conducted in the six papers while Chapter 6 discusses the research findings. Finally, Chapter 7 concludes with a summary of the results, answers the research questions, addresses limitations of the study, and gives suggestions for future research.

Chapter 2

Lean Thinking

Satisfying the rapidly changing customer needs in software development has become an emergent research area as the trend of agile software development addressed above has shown. A correlation between the quality of the process and the quality of the developed software has, however, been a myth, as claimed by Fuggetta (2000). As an answer to the challenge of handling unpredictability accentuated due to uncertainties in requirements and technology, agile project management methodologies (Abrahamsson et al., 2002, 2010) starkly deviate from the conventional project management doctrine. Recently, research has started figuring out whether the Lean way of doing things is the answer for the challenge of managing software development processes of the next generation (Middleton, 2001; Middleton et al., 2005; Middleton and Joyce, 2011). Instead of "Lean way of doing things", this thesis uses the term *Lean approach* to refer to the way of doing things by following Lean thinking.

Using plan-driven methods in software development projects is insufficient for successful execution, mostly because the future (in this case: appropriate software product requirements and designs) is hard to anticipate (Basili and Turner, 1975; Larman and Basili, 2003). In addition, Poppendieck and Poppendieck (2003) anecdotally suggest that development projects lack a systematic way of eliminating waste from the projects. Overall, the Lean thinking is claimed to provide a more concrete way to handle processes than the conventional one (Poppendieck and Poppendieck, 2007). This chapter reviews the background of the Lean thinking from its origin to application of it to the software development. Lean concepts are explained as well. Finally, comparing the Lean approach with the agile one leads us to introduce and study Kanban.

2.1 Origin

Lean thinking can be traced to Toyota (Liker and Hoseus, 2008, 15). A Japanese technique, called *Just-In-Time* (JIT) (Ohno, 1988)[1] (Chapter 2.2.1) originates in the late 1940s and early 1950s when Taiichi Ohno developed *kanbans*[2] in order to control production between processes and in order to implement Just-In-Time manufacturing at Toyota (Gross and McInnis, 2003, 1).

The Statistical Process Control theory (Pfadt and Wheeler, 1995), taught by Edwards Deming in Japan in the 1950s, showed the importance of Quality Assurance (QA). Improved quality improves the throughput of a system (Anders, 2004, 77–94). The Theory of Constraints, introduced in 1984 by Goldratt (1984) provided a way for manufacturing to identify bottlenecks in the production line: the capacity of the weakest link was considered the current system constraint. During the same decade, JIT was catching on in western manufacturing since it offered a way to reduce inventory (Anders, 2004, 13–28). This reduction had a significant effect on the profitability of a manufacturing business (Anders, 2004, 13–28).

Yet another phenomenon of the 1980s was Total Quality Management (TQM). The aim of TQM, in its turn, was to improve quality in order to make manufacturing more profitable. In general, quality improves production because it reduces rework. In the late 1980s, QA and JIT were recognized as a part of the overall improvement of the manufacturing industry. Womack et al. introduced the term "Lean" in 1990 in their book *The Machine That Changed the World*. This book documented the superior Japanese automobile manufacturing processes that were the combination of low inventory of Taiichi Ohno and Shigeo Shingo, and the JIT system with the quality system of Edwards Deming. This combination is known as the *Toyota Production System (TPS)*. More recently, it is called *The Toyota Way* according to Toyota's perception that this management method is applicable outside the production, as well (Anders, 2004, 5–6).

Shingo (1989, 67) reported that TPS, in practice, is 80% waste elimination, 15% production system, and 5% Kanban. Part of the literature (e.g., Anders (2004, 5–6)) considers TPS to be synonymous with the Kanban system. However, this is a misunderstanding that arose since the concept "rules of Kanban" has been interpreted, on one hand, as principles of production but, on the other hand, as Kanban. Kanban from Toyota's

[1] The original, Japanese edition *Toyota seisan hoshiki* by Taiichi Ohno was published in 1978.

[2] I.e., index cards (Poppendieck and Poppendieck, 2007, 10).

viewpoint, regardless, is simply a means of achieving just-in-time (Shingo, 1989, 67). Besides, the term "Lean" differs from the one historically used in Toyota, was accepted within Toyota Motor Sales (in the United States), and then became part of the local culture of that organization (Liker and Hoseus, 2008, 468).

The TPS is probably the most famous example of how to attain success with the aid of the Lean approach. Lean principles are, however, not enough. What is needed is the culture built into the whole corporation. The fourteen principles of doing things in Toyota, called the principles of the Toyota Way, are statements of beliefs and values which are about Toyota's culture. This culture can be encapsulated in four items: (1) long-term philosophy (the purpose and reasons of Toyota's existence), (2) Lean processes (which lead to operational excellence) which emphasize continuous elimination of waste, (3) developing and challenging people and partners through long-term relationships, and, (4) organizational learning driven by problem solving and continuous improvement. Despite its success, the way of doing things in Toyota is always changing (Liker and Hoseus, 2008).

2.2 Key Concepts

This chapter introduces key concepts of Lean thinking. Chapter 2.2.1 introduces the terminology and Chapter 2.2.2 constitutes an overview of the main goal of Lean thinking.

2.2.1 Terminology

Key concepts of Lean thinking are introduced below. In addition to these concepts, The Toyota Way of doing things contains the elements of challenge, respect, and teamwork (Liker and Hoseus, 2008, 14–15). Even though this set of concepts as a whole is too wide to focus on in this thesis, walking the concepts through is necessary in order to understand the ideology and fundamentals behind the Lean approach and its linkage to Lean software development. The concepts excluding *flow*, *Just-In-Time*, *pull*, *value*, and *waste* are Japanese words due to the Japanese origin of Lean thinking.

Andon

The Japanese term *Andon* refers to a system that notifies management, maintenance, and other appropriate workers of a quality or process problem. An electric signboard equipped with signal lights is an example of this kind of system. Andon closely relates to the Jidoka (see below) quality control

method and provides workers with the possibility to stop production in case of abnormality and immediately call for assistance. In Toyota, the first step in Jidoka is identifying problems and each team member must be able and willing to call attention to the problem. The Andon system makes this calling easy for the members (Liker and Hoseus, 2008).

Flow

The English term *flow* in the context refers to the entire value stream, not to productivity of individual steps in the stream. In a pipeline process, for instance, we have to focus the time for ideas to go from beginning to end so that the flow provides the greatest value. The term closely relates to Just-In-Time (see below) (Shalloway et al., 2009, 223–224).

Gemba

By its meaning, "place," applied in manufacturing it is understood as the place where activity really happens, i.e., where the manufacturing work, for example, is done. The term closely relates to Genchi Genbutsu (see below) (Liker and Hoseus, 2008).

Genchi Genbutsu

The Japanese term *Genchi Genbutsu* is one of the key principles of the TPS and means "going to see". In order to understand the full impact of the situation one has to physically go to the place where work is done (Gemba). According to Lean thinking, problems are visible, which makes it sensible to consider Genchi Genbutsu as a key approach in problem solving. It allows management and other observers to see the performance in the manufacturing place where real value is created. Genchi Genbutsu facilitates seeing, for example, whether people are following a repeatable standard process or the material is flowing smoothly through the plant. The concept becomes second nature to Toyota engineers (Liker and Hoseus, 2008).

Hansei

Hansei is a fundamental part of Japanese culture and means self-reflection. The aim is to acknowledge one's own mistakes and to commit to making improvements. The Toyota culture requires executing *Hansei-kan* (a reflection meeting) despite the success level of the project or process executed. The idea is to review what went wrong and what can be improved. In TPS, only "no problem" is a problem during a Hansei-kai and the focus is

on all the deficiencies of both the team and technical processes (Liker and Hoseus, 2008).

Heijunka

Heijunka refers to production smoothing and it aims at reducing Muda (see Section "Muda" in this chapter). The TPS develops the production efficiency by following the Heijunka principle. The general value of utilizing Heijunka is to produce intermediate products at a constant rate and to allow further processing to be carried out at a predictable, constant rate. Toyota applies the principle to its hiring policy as well (Liker and Hoseus, 2008).

Hoshin-kanri

While *Hoshin* means shining metal, compass, or pointing the direction and *Kanri* means management or control, *Hoshin-kanri* translates into "directions and means management" and refers to policy deployment. Regarding TPS, it is a system that sets objectives for improvement. It teaches people to learn how to solve problems and continually improve their work. Hoshin-kanri begins at the very top of Toyota and comes to agreements at each level down to the team member. Every employee has a *hoshin*, defined as particular measurable objectives that are reviewed throughout the year. Hoshin both develops people through the job and increases performance as an organization through teamwork. It interconnects the leadership's vision, values, and philosophies to the daily activity on the Gemba. As a process, Hoshin (1) sets mid-to-long-term management plans and annual Hoshin, (2) prioritizes activities and resources, (3) involves all members in targets or means to achieve them, and (4) maintains the cycle of plan-do-check-act and follow-ups during implementation. By putting all these together, Hoshin-kanri is a mechanism for converting team members' energy into exceptional levels of performance that guides the whole organization in one direction, with the members taking the initiative (Liker and Hoseus, 2008).

Jidoka

Jidoka is considered the other pillar upon which the TPS has been built and is considered to lead to continuous improvement. The TPS understands *Jidoka* as automation with a human touch, often called "intelligent automation". The fundamental idea is that people should not serve machines but vice versa. Jidoka implements rather supervisory functions than production functions. Stop-the-line is an example in case of abnormality.

Jidoka prevents overproduction and producing defective products. By evaluating the process and understanding the reasons behind a problem it can better be ensured that the problem never occurs again (Liker and Hoseus, 2008).

Jishuken

By meaning voluntary self-study, *Jishuken* in the TPS refers to Toyota Kaizen events that are performed with the primary purpose of developing the skills of problem solving and leadership of the managers. Jishuken expects managers to learn and find ways to improve processes. In the sense of self-study, managers may interview team members, identify waste, and make improvements based on their findings. The Jishuken events, are however, driven and led by trained facilitators who are experts in Lean (Liker and Hoseus, 2008).

Just-In-Time (JIT)

The idea of pull-driven JIT is that a production unit does not "push" anything to the customer or unit next to it. This customer or unit rather "pulls" the product needed. Such a policy prevents overproduction: if the customer or unit "pulls" only those products really needed, the producing unit only wastes its time and resources if it produces unnecessary products. Furthermore, when products are ready just in time, inventories become useless since the customer "pulls" the product to himself before the non-value adding inventorying operation (Shingo, 1989).

Womack and Jones (2003, 349) defines the term Just-In-Time as follows: "A system for producing and delivering the right items at the right time in the right amounts."

Kaizen

Kaizen refers to change for the better, either a philosophy or practices that focus upon continuous improvement of processes, which aims at eliminating waste and overly hard work (Muri). In addition, the workplace gets humanized and people learn how to perform. Usually Kaizen concerns all personnel from the chief executive officer to assembly line employees. In the TPS, it is commonly a local improvement with a local area or workgroup related to their own environment and productivity improvements. The line personnel is expected to stop the line when abnormality is revealed. Then, tracing the reason for this abnormality leads to an improvement suggestion

that eventually may initiate a Kaizen. The essence of Kaizen is the notion that line workers, engineers, and managers collaborate continually to identify incremental changes and systematize production tasks in order to establish a smoother flow (Liker and Hoseus, 2008).

Kaikaku

Kaikaku means radical improvement within a limited time, in contrast to Kaizen. The aim is to eliminate Muda. In TPS, such a radical improvement may occur due to introducing new production techniques or equipment, strategies, or knowledge. Typically it is initiated by management but can also be launched by external factors, such as market conditions (Womack and Jones, 2003).

Kanban

The Japanese word Kanban refers to a signboard. When the term is used in manufacturing, it means a scheduling system that hints what, when, and how much to produce. Toyota, for example, has successfully applied Kanban in practice as one part of TPS resulting in a way for promoting improvements (Hiranabe, 2008).

Kanban is basically a flow control mechanism for pull-driven Just-In-Time production in which the upstream processing activities are triggered by the downstream process demand signals. In general, Kanban has three rules: (1) visualize the workflow, (2) limit work in progress (WIP) at each workflow state, and (3) measure the cycle-time, i.e., average time to complete one item (Kniberg, 2009).

Kanban does not intervene in management despite its importance, i.e., how to do things. Instead, it is inclusive of management. In other words, management is involved and it is committed to abide by the methods the teams have selected to do their work. In addition, management is part of discussions about how the work is being tracked and performed. Kanban combines defining and managing a workflow: this workflow based on queues and control loops is managed by limiting WIPs (Shalloway et al., 2009, 98–100).

Muda

Muda for waste means activity that is wasteful and does not add value or is unproductive (Ohno, 1988; Shingo, 1989). Womack and Jones (2003, 43) divide wasteful activity that has occurred along the value stream into two

types: steps that create no value but are unavoidable (type one Muda) and steps that create no value and are immediately avoidable (type two Muda).

Mura

Mura refers to inconsistency in physical matter or the human spiritual condition, and to unevenness. Uneven workloads is an example of Mura (Ohno, 1988; Shingo, 1989).

Muri

Muri means unreasonableness, overburden, or absurdity (Ohno, 1988).

Pull

In the *pull* method, the next process withdraws the quantities it requires from the preceding process (Ohno, 1988). See Section "Just-In-Time (JIT)" in this chapter.

Value

Oppenheim et al. (2010) define the *value* for the present purpose as follows:

> "Value is defined as the delivery of a complex system satisfying all stakeholders, which implies a flawless product or mission delivered with at minimum cost, in the shortest possible schedule, fully satisfying the customer and other stakeholders during the product or mission lifecycle."

In this sense, value-added activity satisfies the following three conditions: (1) transform information or material or reduce uncertainty, (2) the customer is willing to pay for it, and (3) it is done right the first time (Oppenheim et al., 2010).

Waste

The English word *waste* in the sense of Lean thinking refers to the Japanese word Muda (see Section "Muda" in this chapter) (Ohno, 1988).

Yokoten

To *Yokoten* is to spread across or propagate. In nature, Yokoten is multiplying of grafts and saplings of a large tree into many new trees. While each new tree will grow differently in its separate, unique way, it will thrive

with properly prepared weather and soil conditions as will its peers. Correspondingly in the TPS, Yokoten is not cloning nor copying but improving what has been seen. Toyota's Kaizen process includes Yokoten (Liker and Hoseus, 2008).

2.2.2 Primary Goal in Lean Thinking

The Lean concepts presented in Chapter 2.2.1 have a common goal: producing value. Lean thinking emphasizes value-adding production. The receiver of this value should be end users, customers, or other interested parties. In order to produce value, waste elimination is considered as the most important principle of Lean thinking (Poppendieck and Poppendieck, 2003).

Mandić et al. (2010), however, argue that defining a value in the context of software engineering is difficult and complex. Shalloway et al. (2009, 14–15) suggest that a primary goal of Lean thinking should be optimizing the whole with sustainability and speed. This suggestion can be considered a practical way how to produce value, i.e., how to reach that goal.

As an example, let us think of a feature of software being produced in a software development project. Once this feature has been implemented, it should be tested. If the production flow is uneven (Mura), it may be that some features are produced faster than they are tested. In such a case, the testing phase becomes a bottleneck resulting in overburdening (Muri). In order to accelerate the production flow and prevent Muri, other implementers should assist those who are testing software. As a result, the flow gets smoother and the acceleration benefits the integration team who does not have to wait without doing anything. Waiting is considered waste since it does not add any value for the customer. In other words, waiting is wasteful activity (type two Muda). This example demonstrates how value can be produced by optimizing the whole and by focusing on the flow.

Since Lean thinking addresses waste elimination, the general-level terms of Muda, Mura, and Muri (Chapter 2.2.1) have been refined into more specific pieces in software development as has been done in manufacturing. Shingo (1989) identifies seven types of manufacturing waste. Table 2.1 translates these items into software development.

Recognizing waste, however, can depend on the context. Emiliani et al. (2005), based on Emiliani et al. (2003), extend the list of waste with *behaviors* that are recognized by senior managers in a Lean management system. Meanwhile, Oppenheim (2004) summarizes the work of Millard[3] wherein the words further originated from the manufacturing waste are redefined

[3]R.L. Millard, *Value stream analysis and mapping for product development.* Master's

Manufacturing Waste (Shingo, 1989)	**Software Development Waste** (Poppendieck and Poppendieck, 2007)
[In-process] inventory	Partially done work
Over-production	Extra features
Extra processing	Relearning
Transportation	Handoffs
Motion	Task switching
Waiting	Delays
Defects	Defects

Table 2.1: The seven kinds of manufacturing waste (Shingo, 1989, 191) translated into software development by Poppendieck and Poppendieck (2007, 73–74).

in the context of product development as follows: Inventory means keeping more information than needed. Over-production refers to creating unnecessary information while over-processing (extra processing) is about working more than necessary to produce the outcome. Transportation, in its turn, relates to inefficient transmittal of information. Unnecessary movement (motion) is about people having to move to gain or access information. Waiting means waiting for information, inputs, approvals, and releases, among others. Finally, defects refer to insufficient quality of information and requiring rework.

Nevertheless, in the area of software development, Poppendieck and Poppendieck (2007, 73–82) suggest a classification of waste as follows:

- *Partially done work*, "inventory of software development" does not guarantee that it works before completed, tested, and integrated. It is not guaranteed that partially done work really solves the customer's problem either. Instead, it ties up resources.

- *Extra features* consume resources when tracked, compiled, integrated, and tested. The more of these non-value adding "just-in-case" features, the more complexity and potential defects there are.

- *Relearning* wastes resources and adds no value for the customer. Rediscovering a known but forgotten thing is rework. Ignoring knowledge that people bring to the workplace, in its turn, destroys utilizing their potential.

thesis in Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2001.

- *Handoffs* leave the major part of knowledge behind in the mind of originators. Tacit knowledge is difficult to transport to other people through documentation. In brief, documents cannot contain all of the information that the other people in line need to know.

- *Task switching* between jobs or tasks takes time, much because of re-orientation and re-focusing. In addition, lack of immediate access to other developers and other representatives disrupts concentration.

- *Delays* slow down realizing value for the customer. Waiting for people to be available who are being busy in other areas causes waste. Critical decision-making in developing requires a good understanding of the situation from the developer. Moreover, it requires someone with knowledge in the room to answer the remaining questions. Lack of this understanding and knowledge results in a new decision problem: should the developer stop in order to try to find out the answer, switch to another task, or just make a guess without stopping.

- *Defects* in the code take resources to fix them. Defects, even small ones, revealed after weeks are typically more serious problems than big defects found immediately.

This list of waste, summarized in Table 2.1, is a revision from Poppendieck and Poppendieck (2003) where the translation of waste from manufacturing to software development was introduced. The list does not concern management but rather the development process of software. Yet, we have not found feedback that could indicate its validity. Despite its lack of scientific evidence, it is based on Lean thinking and experience of practice of its developers.

2.3 Implementation Practices

The way of attaining value by following the Lean approach differs from area to area where it is applied to, as stated in Chapter 2.2.2. The following sections highlight such differences and provide examples.

2.3.1 Lean Manufacturing vs. Lean Development

Lean thinking has been adopted and successfully applied to a variety of manufacturing areas from wood production (e.g., Cumbo et al. (2006)) to car manufacturing (e.g., Liker and Hoseus (2008)). Cumbo et al. (2006),

among others, state that the quality of products and the efficiency of product development in manufacturing have increased due to the Lean approach.

Transforming successfully from traditional ways to Lean, however, has appeared to be problematic as shown below. Such problems are, for example, obtaining open office space to locate teams together, gaining executive support, and training and informing people to diminish resistance of change (Parnell-Klabo, 2006). In that case study of Parnell-Klabo (2006), after removing such obstacles, the lead-time for delivery was decreased by 40% to 50%. Karlsson and Åhlström (1996) mention more problems: First, creating a cross-functional focus is difficult when people feel comfortable with their current function. Second, simultaneous engineering is unfamiliar for people coming from sequential processes. Third, project coordination is problematic due to misunderstandings of the unfamiliar work disciplines of others. Moreover, managing a vision-based organization may not be fluent when people are used to detailed specifications and instructions. Finally, expecting cost estimations in a highly flexible product development process hinders the relationship to customers. In the organizational context, even getting familiarized with Lean thinking has been challenging for organizations (Liker and Hoseus, 2008, 27–31) due to the holistic entity of Lean (Chapter 2.1). Typically, the Lean approach requires redefining the work of functions, departments, and companies in order to enable a positive contribution to value creation (Womack and Jones, 2003, 24).

Regarding management and processes, Koskela (2000, 187–189) identifies the following seven types of preconditions for a sound process: (1) construction design, (2) components and materials, (3) workers, (4) equipment, (5) space, (6) connection works, and (7) external conditions. Bertelsen et al. (2006) stress that in thesense of production management, the seven types of preconditions of Koskela (2000) cannot be managed satisfactory with a single management method. Regardless, there is no orthodox Lean approach as shown by the overlapping Lean principles reported in the literature (Chapter 2.3.2). Middleton and Joyce (2011), among others, agree on this lack of orthodoxy.

2.3.2 Principles

Nowadays, Lean thinking has extended from Lean manufacturing to Lean enterprise. The operational areas include Lean software development. Similar to the non-problematic free transforming to Lean described in Chapter 2.3.1, translating Lean practices from manufacturing and supply chain management to software development has, in its turn, encountered challenges due to differences of operations and logistics to software and devel-

opment (Poppendieck and Poppendieck, 2007, 11–17).

The Lean principles presented in the literature vary depending on their implementation context. Womack and Jones (2003, 16–26), for example, present the five following Lean principles regarding Lean production:

- specify value
- identify the value stream
- create the flow
- establish the pull
- seek perfection.

Womack and Jones (2003) emphasize that customer value is created by producers. For this reason, customers need producers. Henry Ford managed to demonstrate the potential of flow with the assembly line of the Model T Ford. Oppenheim (2004) states that the success of this assembly line was the ability to split the complex craftwork into separate tasks of short and equal duration. The real challenge of the continuous flow according to (Ohno, 1988), however, was reaching the flow in low-production without expensive assembly lines and, moreover, learning to rapidly change over tools from one product to the next. In order to reduce inventories, customers should "pull" products from producers as needed rather than producers "push" unwanted products onto customers (Womack and Jones, 2003, 16–26).

Lean consumption, instead, relates to trade between the consumers and the providers and it consists of the following six principles (Womack and Jones, 2005):

- solve the customer's problem completely by insuring that all the goods and services work, and work together
- do not waste the customer's time
- provide exactly what the customer wants
- provide what's wanted exactly where it's wanted
- provide what is wanted where it is wanted exactly when it is wanted
- continually aggregate solutions to reduce the customer's time and hassle.

The principles of Lean consumption closely relate to the principles of Lean production. Customers on the entertainment electronics market, for example, want everything (hardware, software, and support services) to work together reliably and seamlessly. In such a complex process, things are not likely to flow smoothly because of the providers' inability to work together to perfect the whole consumption process. This leads to the first principle. Lean consumption principles suggest a holistic way, in addition to solving the customer's specific problem, to identify the systemic source of the problem in order to fix it entirely. In contrast to the providers' culture of wasting the customers' time, Lean providers look at the problem from the customer's viewpoint and *prevent wasting customer's time.* Furthermore, *providing exactly what the customer wants* emphasizes "pull". The speed of stock replenishment systems helps to restock precisely what a customer has just pulled off the shelf. Providing *what is wanted exactly where it is wanted* completes the previous principle. Lean logistics techniques make it possible to offer a wide range of formats with uniform pricing and, meanwhile, to serve every customer need (Womack and Jones, 2005).

The principle of *when it is wanted* needs aligning with the complex provision streams of multiple companies. This aligning would enable conditions for customers wherein they share their plans with a producer and order the product in advance, resulting in a customized product for a reduced price. Such a purchasing model has already shown its potential in services like vacation cruises. Finally, *continuous aggregation of solutions to reduce the customer's time and hassle* is possible by reducing supply bases for items and by involving fewer suppliers with deeper knowledge, as Toyota does (Womack and Jones, 2005).

Meanwhile, Nightingale (2009) introduces seven principles related to Lean enterprise thinking:

- adopt a holistic approach to enterprise transformation
- identify relevant stakeholders and determine their value proposition
- focus on enterprise effectiveness before efficiency
- address internal and external enterprise interdependencies
- ensure stability and flow both within and across the enterprise
- cultivate leadership to support and drive enterprise behaviors
- emphasize organizational learning.

Even though the viewpoint lies in transformation, Lean principles are present. A need for a holistic approach is highlighted through multiple instances. Focusing only on some areas cannot make the process flow. Multiple stakeholders, such as customers, suppliers, partners, and employees, find different benefits, rewards, or worth for their contributions. Hence, the enterprise has to distribute value to all of them. Enterprise effectiveness needs focus before efficiency. The enterprise strategic objectives, the resources to produce value, and the value delivering mechanism must be understood before optimization. In other words, "doing the right thing" comes before "doing it right" (Nightingale, 2009).

The principle of internal and external enterprise interdependencies should be addressed from the three aspects: what the enterprise can control, what the enterprise influences, and what the constraints on the enterprise are. Stability of value delivery helps to determine the enterprise's current state thus providing a baseline for improvement. Flow of value, instead, helps to focus on improving the value delivery to the key stakeholders. Leadership must be cultivated at all three levels: senior, middle management, and grass-root (i.e., operational) level. When these levels have been aligned, the transformation effort is expected to become self-sustaining. Finally, the principle of emphasizing organizational learning should gain knowledge about the organization's processes and value-creating manners (Nightingale, 2009).

In order to adopt Lean principles to software development, translations that take into account the nature of software engineering, have been suggested as follows (Poppendieck and Poppendieck, 2007):

- eliminate waste
- build quality in
- create knowledge
- defer commitment
- deliver fast
- respect people
- optimize the whole.

These principles are the revision of the original wording of Poppendieck and Poppendieck (2003). *Eliminating waste*, i.e., eliminating activity that is non-value-adding, is the primary principle of Lean production. Lean

software development has the same focus but the principle attempts to shorten the timeline by removing such waste. The elimination is not possible without recognizing waste (Table 2.1). This recognizing, on its part, requires a deep understanding of what the value is for a certain customer (Poppendieck and Poppendieck, 2007).

Building quality in (formerly build integrity in) is another goal of the development from the outset rather than testing the quality later. Queues, such as defect tracking systems, are queues of rework, partially done work. *Create knowledge* (formerly amplify learning), in its turn, addresses that software development is a knowledge-creating process. Even a detailed design document written ahead does not prevent the fact that the detailed design of software happens during coding. Locking down all guesses and anticipations prematurely is a waste of time (Poppendieck and Poppendieck, 2007, 23–41).

Lean organizations should be aware that problems are always a part of a complex environment and that they have to improve their processes continually. Each abnormality should launch searching for the root cause of the problem in order to eliminate it. *Deferring commitment* (formerly decide as late as possible) is based on the fact that the more that is known about the situation being decided, the less uncertainty there is and less guesses have to be made. For this reason, locking critical design decisions that will be difficult to change should be avoided: irreversible decisions should be made as late as possible. The principle of *delivering fast* (formerly deliver as fast as possible) prevents customers from changing their minds because they do not have time to do so. Competing on the basis of time typically brings cost advantages over the competitors when waste including defects that cost money, has been minimized. The principle of *build quality in*, mentioned above, enables sustaining high speed. High speed is not, however, synonymous with botching up (Poppendieck and Poppendieck, 2007, 23–41).

Respecting people (formerly empower the team) also has a salient position in the *Toyota Product Development system*[4]. Three of the four cornerstones of that system relate to people as follows: (1) A people-respecting company develops good leaders, which often relates to successful products. (2) Expert technical workforce is required to maintain a competitive advantage in a certain area. (3) Teams are trusted to self-organize to meet the goals given; instead of telling people what and how to do, people figure this out by themselves (Morgan and Liker, 2006).

The final principle, *optimize the whole* (formerly see the whole), refers

[4]Not to be confused with Toyota Production System.

to the whole value stream from receiving an order to addressing a customer need. Serious delays can usually be tracked to handoffs of responsibility wherein the customers' concerns have been forgotten (Poppendieck and Poppendieck, 2007, 23–41).

The Lean principle of optimizing the whole closely relates to the concept of flow. Flow is a fundamental part of Lean thinking as demonstrated above. By focusing on flow, waste can be revealed in software development as well (Reinertsen, 2009, 3–21). Oppenheim (2004) concludes that Lean principles that have yielded benefits in production applications are valid in product development, as well: work should be organized as an uninterrupted flow that proceeds steadily through all processes without rework or backflow.

2.3.3 Applying in Software Development

Applying the Lean principles presented in Chapter 2.3.2 into software development appears to be reasonable for, at least, two reasons. First, Lean thinking takes into account drawbacks of the plan-driven production, as presented above. The plan-driven production has focused on planning everything from the outset of a project and is considered to be a heavily documentation-based way (Sommerville, 2007). The waterfall model (Royce, 1970) is an example of this way where the process is executed sequentially step-by-step through different software development disciplines, such as requirement engineering, architecture design, and implementation. Even though the original suggestion of Royce (1970) was to walk the process through twice, the model still assumes that requirements are relatively stable. Hence, lead-times have been allowed to be long. Second, software markets today are highly dynamic, which forces producers to respond rapidly to changes (Poppendieck and Poppendieck, 2007). Currently, the software industry is expected to answer to this challenge of dynamicity.

The Lean approach provides advantages for industry sectors, such as buffering in schedules, close cooperation with customers enabling reception of feedback, and regularizing face-to-face meetings of managers (Karlsson and Åhlström, 1996). It is realistic to expect reaching these advantages also in software development, namely, Lean principles applied to different sectors (Chapter 2.3.2) appear to have an interrelationship with Lean thinking: these principles address such important concepts of the TPS as Hansei, Heijunka, Jidoka, Jishuken, JIT, Kaizen, and Muda (Chapter 2.2.1) from the viewpoint of producing value. Despite the different viewpoints of Lean manufacturing and Lean development addressed by the literature in Chapter 2.3.1, similarities exist as well (Table 2.2).

A small amount of research done, such as Morgan (1998), reports a

Lean Manufacturing	**Lean Development**
Frequent set-up changes	Frequent product changes (software releases)
Short manufacturing throughput time	Short development time
Reduced work-in-process inventory between manufacturing steps	Reduced information inventory between development steps
Frequent transfer of small batches of parts between manufacturing steps	Frequent transfer of preliminary information between development steps
Reduced inventory requires slack resources and more information flow between steps	Reduced development time requires slack resources and information flow between stages
Adaptability to changes in volume, product mix, and product design	Adaptability to changes in product design, schedule, and cost targets
Broad assignments for production workers gives higher productivity	Broad task assignments for engineers (developers) gives higher productivity
Focus on quick problem solving and continuous process improvement	Focus on frequent incremental innovation and continuous product and process improvement
Simultaneous improvement in quality, delivery time, and manufacturing productivity	Simultaneous improvement in quality, development time, and development productivity

Table 2.2: Similarities between Lean manufacturing and effective product development (Poppendieck and Poppendieck, 2007, 14).

valuable potential of Lean thinking in the sense of improving the performance of software development projects. Regardless of this potential, Lean thinking has not been widely applied to software development.

In a case study of Middleton et al. (2005), a majority of that company that was studied, agreed that the Lean approach is applicable to software engineering. Further, the statistics showed a 25% gain in productivity. Schedule slippages of months or years were reduced to four weeks. Meanwhile, the time to fix defects was reduced by 65% to 80%. Finally, the product release using the Lean approach exceeded the expectations of the customer. In that case project, the team members applied 11 Lean principles or techniques to their software development. These were: (1) continuous-flow processing, (2) customer-defined value, (3) design structure matrix and flow, (4) common tempo or "takt" time, (5) linked processes, (6) standardized procedures, (7) eliminate rework, (8) balancing workloads, (9) posting results, (10) data-driven decisions, and (11) minimize inventory.

Moreover, the use of the Lean approach has been reported to benefit capstone software projects. In an experiment of Perera and Fernando (2007), for example, a hybrid process of agile and Lean produced more lines of code than agile. At the beginning, the group that used the hybrid process also found more defects than the group that used the agile process. The reasons were autonomous and value perfection norms, and paying more attention to the perfection than the agile group. At the later stages the situation reversed, i.e., the hybrid group started to get a stable minimal defect rate while the agile group experienced high and varying defect rates. This was a consequence from the difference of the amount of hidden unfixed defects in early developments. Therefore, fewer defects in later stages helped to stabilize the project schedule and to reduce costs caused by defect fixing.

Further improvements may be achieved in disciplines such as product planning and people management. Extending the viewpoint from car manufacturing to these disciplines attempts to make Lean thinking even more relevant for software engineering than the manufacturing: improving software development needs focusing on the overall development cycle (Petersen, 2010, 40). Outside software development, this focus is already being utilized to make the process more successful. The Toyota Product Development system, for example, extends the Lean manufacturing to the Lean product development wherein the design, manufacturing, and human resource management are focused and integrated together (Morgan and Liker, 2006). To improve the performance of software development projects, particularly the human aspect is a key as will be addressed in

Chapter 3.1. This, in its turn, gives us reason to think that Lean thinking is worth applying to software development.

2.4 Relation to Agile Software Development

This chapter compares the Lean approach with agile practices in software development. Agile methods (Abrahamsson et al., 2002) (1) are widely used and (2) have been considered to provide a solution for many problems derived from the plan-driven, conventional software development method (Larman and Basili, 2003; Ramasubbu and Balan, 2009; Takeuchi and Nonaka, 1986). Thereby, the motivation for the comparison is to find out what additional value the Lean approach brings to software development.

The agile software development methods have improved the output of the plan-driven methods in software developing projects in the 1990s and 2000s (Díaz et al., 2009; Sutherland et al., 2009). In contrast to the plan-driven methods, the agile approach consists of the four following values cited from the Agile Manifesto (Beck et al., 2001):

(1) Individuals and interactions over processes and tools.

(2) Working software over comprehensive documentation.

(3) Customer collaboration over contract negation.

(4) Responding to change over following the plan.

These values are based on the 12 principles of the agile approach which concisely are as follows (Beck et al., 2001):

(i) Satisfy the customer through early and continuous delivery of valuable software.

(ii) Welcome changing requirements.

(iii) Deliver working software frequently.

(iv) Business people and developers must work together daily throughout the project.

(v) Build projects around motivated individuals.

(vi) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

(vii) Working software is the primary measure of progress.

(viii) Agile processes promote sustainable development.

(ix) Continuous attention to technical excellence and good design enhances agility.

(x) Simplicity – the art of maximizing the amount of work not done – is essential.

(xi) The best architectures, requirements, and designs emerge from self-organizing teams.

(xii) At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly.

The agile approach implements the Lean basic concepts in software development and emphasizes the customer satisfaction and the continuous improvement of the development process (Sillitti and Succi, 2006, 2008). The agile approach attempts to deliver working software continuously, which enables checking by demonstrations whether the customer needs are fulfilled (Petersen, 2010, 41–42, 183). The overall goal of Lean development, instead, is to reach a smooth, continuous flow of production with a minimum waste and maximum flexibility in the process (Oppenheim, 2004). Petersen (2010, 53–70) compares practices of the agile methods with Lean. He recognizes totally 26 principles and groups them into the following seven categories: requirements engineering, design and implementation, quality assurance, software releases, project planning, team management, and end-to-end flow. Table 2.3 summarizes the comparison.

According to Table 2.3, both the advantages and disadvantages of both approaches are evident. Permission to ignore the on-site customer (P01) in Lean software engineering, for instance, allows freedom and it may lower the pressure of the team when the customer is not located at the development site. The lack of demand of Coding standards (P04) and Team code-ownership (P05) in the Lean approach, again, allows freedom but may lead to various ways of writing and structuring code inside the team. Low dependency architecture (P06), instead, is connected to the Lean approach only, which allows developing the components independently (i.e., the deliveries of the components are not dependent on each other). The planning game (P17) is unique to the agile approach, which makes it more mechanical and formal than the Lean approach. More freedom can be attained in the Lean approach by ignoring the 40-hour week (P20) and Stand-up meeting (P21).

In contrast, Value-stream mapping (P23) emphasizes the idea of Lean software engineering since the purpose of processes is to create value for

Group		Used in Agile SE	Used in Lean SE
Requirements Practices			
P01	On-site customer	x	
P02	Metaphors/Stories	x	x
Design and Implementation Practices			
P03	Refactoring	x	x
P04	Coding standards	x	
P05	Team code-ownership	x	
P06	Low dependency architecture		x
Quality Assurance Practices			
P07	Test-driven development and test automation	x	x
P08	Pair-programming	x	x
P09	Continuous integration	x	x
P10	Reviews and inspections	x	x
P11	Configuration management	x	x
Software Release Practices			
P12	Incremental deliveries to the customer	x	x
P13	Separation between internal and external releases	x	x
Planning Practices			
P14	Short iterations	x	x
P15	Adaptive planning with highest priority user stories/requirements	x	x
P16	Time-boxing	x	x
P17	The planning game	x	
Team Management Practices			
P18	Co-located development	x	x
P19	Cross-functional teams	x	x
P20	40-hour week	x	
P21	Stand-up meeting	x	
P22	Team chooses own tasks	x	x
End-to-end Flow Practices			
P23	Value-stream mapping		x
P24	Inventory management with queuing theory and theory of constraints		x
P25	Chief Engineer		x
P26	Kanban pull-system		x

Table 2.3: Summary of comparison of Petersen (2010, 41–70) between agile and Lean software engineering (SE) practices.

customers. Inventory management with queuing theory and theory of constraints (P24), on its part, executes the Lean principle of eliminating waste. Inventory, among others, is considered waste (Chapter 2.2.2). Chief Engineer (P25) relates to optimizing value creation and the pull system (P26), as stated in Chapter 2.2, is unique for Kanban that is described in Chapter 2.5.

Overall, the practices of quality assurance and software release are included in both approaches. Fifteen of the practices mentioned in Table 2.3 are shared between agile and Lean. The following six practices are considered to be unique to agile methods: on-site customer, coding standards, team-code ownership, planning game, 40-hour week, and stand-up meetings. Even though Lean is not agile in this sense, all agile practices identified support Lean principles. The seventh Lean principle (i.e., "optimize the whole") distinguishes the Lean approach from the agile one. From the process viewpoint, the Lean approach does not propose, in contrast to agile, a workflow or the production of specific artifacts. Instead, it states principles and provides analysis tools for processes. This focus guides to improve processes to achieve a good flow of value. Such process freedom makes Lean more generally applicable than agile methods (Petersen, 2010, 41–70).

The comparison of Petersen (2010) shows similarities and differences from the seven following aspects: people management and leadership, quality of the product, release of the product, flexibility, priority of the customer needs and values, learning, and end-to-end flow (Table 2.3). Based on these aspects, Petersen (2010) concludes that the Lean approach includes all the principles of agile methods but not vice versa: the agile principles do not emphasize the end-to-end focus on flow. According to this observation, Lean appears to complement agile methods by the flow aspect and by concretizing in the form of seven kinds of waste what does not provide the value for the customer. In addition, Middleton and Joyce (2011) remind us that agile-based Scrum, for example, is, in essence, a batch model of "push" while Lean has the pull mechanism.

As a conclusion, utilizing the Lean approach successfully seems to require experience from its users. The Agile Manifesto contains more principles than has been presented in Lean software development, which appears to make the approach more mechanical than Lean. The agile principles seem to be more descriptive than Lean ones. Thereby, it may be easier for teams to adapt them into practice than Lean principles. Sillitti and Succi (2008), however, conclude that the agile approach does not pretend to be useful for any kind of software project or organization. The Lean approach,

by stating principles and providing analysis tools for processes, guides its users to improve processes to achieve a good flow of value, which has been claimed to make Lean more generally applicable than agile methods (Petersen, 2010, 41–70). Petersen (2010, 217) concludes that the end-to-end perspective of the whole value flow through development distinguishes Lean from agile. Shalloway et al. (2009, 6–23), on their part, summarize that Lean extends agile by providing guidance for agile practices in new situations, telling us to focus on time of development instead of resources utilized, and reminding us to optimize the whole instead of trying to get each step done in the most efficient way.

2.5 Kanban in Practice

The literature addressed in Chapter 2.4 gave the motivation for why one should utilize the Lean approach in software development. In order to perform software development by following Lean principles, a need for an appropriate software process model arises. Hence, this chapter introduces Kanban.

The Kanban process model (Gross and McInnis, 2003) executes Lean thinking, including "pull", in practice (Becker and Szczerbicka, 1998; Chai, 2008). Moreover, it is one of the key operation management tools in Lean manufacturing (Liker, 2004, 176) and provides a way to prevent Muda, Mura, and Muri (Ohno, 1988). In addition to value stream mapping (Mujtaba et al., 2010), inventory management (Anders, 2004) containing the queuing theory (Poppendieck and Poppendieck, 2007, 100–114), and the theory of constraints (Goldratt, 1984), the end-to-end focus is supported by pull systems (Gross and McInnis, 2003). Value stream mapping shows processing and waiting times by visualizing the development life-cycle (Mujtaba et al., 2010). Inventory management, instead, suggests limiting the simultaneous work in progress: the goal is to minimize partially done work and task switching that are considered waste (Chapter 2.4). Pull systems, on their parts, aim at preventing overload of the development process (Chapter 2.2.1).

In manufacturing operations, the different kinds of waste are basically straightforward to detect by observing the physical material flows and machine or worker activities (Goodson, 2002); (Lui and Chan, 2008, 3–11). Invisibility of waste in software development, however, restrains the progress (Poppendieck and Poppendieck, 2003). Partially done work, for example, involves such emerging issues as requirement and error inventories. In order to manage these inventories, extra processes (i.e., waste) may be born.

Benefits of Kanban scheduling are reduced inventory, improved flow, prevented overproduction, operations-level control, visualized schedule and management of the process, improved responsiveness to changes in demand, minimized risks of inventory obsolescence, and increased ability to manage the supply change (Gross and McInnis, 2003).

In contrast to Scrum, for example, Kanban does not require breaking down features into stories in order to make these stories fit into an artificial deadline caused by a time-boxed iteration scheme. Instead, Kanban requires that a workflow the team creates has to contain explicitly defined rules and limits. This helps teams move their focus from blaming individuals to the process (Shalloway et al., 2009, 100).

From the human aspect, Kotter (1996, 3–16) and Watson (2002, 437–448) argue that without directive behavior (i.e., management) and supportive behavior (i.e., leadership), a staff head cannot reach goals well enough. Moreover, Howell et al. (2004) argue that the theoretical foundation that connects leadership and people aspects holistically is still missing. Kanban, as a method, empowers people with a minimum set of required rules to follow, and, is often presumed to attain the flow (Shinkle, 2009; Ladas, 2009).

While Kanban enables clarifying the workers' awareness of the current production issues and forthcoming tasks, it does not recommend any particular project phases, milestones, or partitioning tasks. Due to this liberty, it is up to a project team to build and customize the appropriate practices for its project. When successful, the impact on the project is supposed to be positive. Furthermore, despite the demand to visualize the workflow, there are no particular rules concerning how to implement the content of the Kanban board. In their basic form in production environments, Kanban controls are typically implemented with physical index cards (usually called *tickets*) moving along with the material. The cards then act as the flow-control tickets between the different work stations or processes (Kniberg, 2009).

Figure 2.1 illustrates one realization of Kanban as a table (such as a wall-paper with sticky notes). Each project task (card) flows from one state to another (from left to right) as it progresses. Hence, the overall project situation can be seen at a glance while the dynamic moving of the task cards indicates the project progress (or blocking) over time. The numbers 6, 2, and 2 represent the WIP limits for their columns meaning that no more than the announced number of tickets is allowed to be located in that column simultaneously.

Some practices, regardless, have been suggested in the literature. Ladas

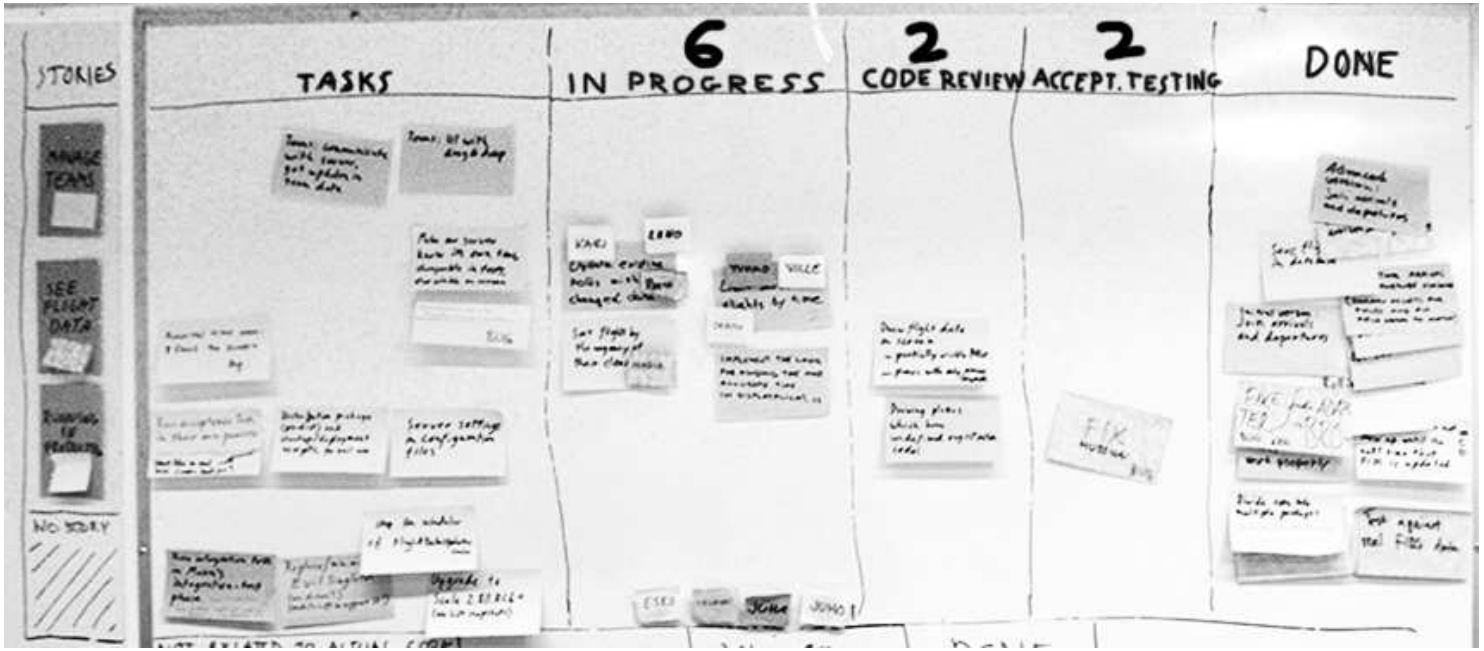

Figure 2.1: An example of one implementation of the Kanban board used in one of our case projects.

(2009), for example, suggests that the amount of tasks in progress simultaneously (i.e., WIP) should be adjusted to the reasonable capacity in use. Middleton (2001) claims that this amount should be minimized in order to keep a high quality. Shinkle (2009), instead, argues that minimizing this amount is not the best solution. Project flow stages wherein the tasks progress from stage to stage has also been suggested. If each stage, such as *To Do*, *Planning*, *Design*, and *Coding*, gets its own *Ready* stage, the blockages in the workflow become more visible (Ladas, 2009). Alwardt et al. (2009) state that tasks should be prioritized. Moreover, laborious tasks should be partitioned before setting them as assigned (Shinkle, 2009). In more general, Gross and McInnis (2003, 8–13) suggest the following seven steps to implementing Kanban: (1) Conduct data collection in order to characterize the production or development process, (2) calculate the Kanban size, which will utilize the production requirements among others, (3) design Kanban so it answers the question of how Kanban will be implemented, (4) train everyone, (5) start Kanban, (6) audit and maintain Kanban, and (7) improve Kanban.

The properties and effects of the original Kanban concept in the TPS are as follows (Hiranabe, 2008).

- Physical: Physical cards are located on, for example, a plain whiteboard. In Figure 2.1, for instance, the tickets on the table represent tasks or features. Those tasks being carried out have been marked with the names of the corresponding developers. This way allows people to see the status of the progress at a glance.

- Limiting WIP: Kanban limits WIP in order to prevent Muri. For example, in the "Code Review" column in Figure 2.1, the WIP number has been set at two. This WIP limit means that no more than two tickets are allowed to be in the column simultaneously. If a task being code reviewed was problematic, carrying it out without the assistance of others would restrain the flow. Other people, once they have finished their tasks, cannot "pull" another ticket into the column because it is already full (due to the WIP limit). They rather have to help with the problematic ticket in order to free some space in the column for new tickets. In this way, the flow is supposed to be smooth and bottlenecks avoidable. Overall, defining WIP limits for each activity, the average cycle time can be minimized.

- Continuous flow: Kanban notifies about the needs of production before the store runs out of stock. In addition to avoiding bottlenecks (see the previous bullet), a smooth flow also means that a customer

or the next unit in the chain (a team, or a member of a team, for example) does not have to wait for the outputs of the process of the previous unit.

- Pull: The downstream process "pulls" items from the upstream process. As explained in Chapter 2.2.1, pull-driven actions closely relate to JIT and to keeping things flowing.
- Self-directing: Kanban has all information on what to do and makes production autonomous in a non-centralized manner and without micro-management. As stated, people can see the status of the progress at a glance (as an example, see Figure 2.1) and determine whether bottlenecks or starvation exist.
- Visual: Kanban is stacked or posted to show the current status and progress visually (Figure 2.1).
- Signal: The visual status of Kanban signals the next withdrawal or production actions.
- Kaizen: The visual process flow informs and stimulates Kaizen.

Moreover, (Shalloway et al., 2009, 100–101) argue that Kanban (1) diminishes the fear of committing to estimations of per-stories, (2) highlights the team's performance over individuals', which makes it rather a team process than one for individuals, (3) focuses on improving the workflow process, (4) allows reflection about concrete measures, and (5) as a transparent process, allows involving management in Kaizen.

In spite of the potential in manufacturing, there are only a few studies regarding how Kanban fits in software development projects. In contrast to the pipeline (manufacturing), a project is, according to PMI (2008, 1) "a temporary endeavor undertaken to create a unique product, service, or result". Translation is needed as it was needed when applying principles of Lean thinking to software development (Chapter 2.3).

Kanban in software engineering is based on the following beliefs: (1) software development is about managing and creating knowledge, (2) software development processes can be managed and described in terms of queues and control loops accordingly, and (3) some representation of information that flows through the system is required (Shalloway et al., 2009, 96).

Basic principles of Kanban-based management can be applied to software production functions by reinterpreting the concept of physical material. In software development, the material flows are replaced by informa-

tion flows. In general, the WIP items then represent the various work tasks in the software development projects (Hiranabe, 2008).

After all, the use of Kanban is not a method per se. The concept of Kanban is used to justify the triggering of activities. In other words, it puts the idea of "pull" into practice. In the context of software development, Kanban is used as an instrument to organize the activities within a team (Janes and Succi, 2009).

2.6 Summary

Chapter 2 introduced fundamental concepts of Lean thinking and reviewed its origin and its practical application. While the concept of Lean thinking is not new, software development has not applied it until recently. As a conclusion, the literature suggests that Lean thinking has potential for software engineering. Kanban as a part of Lean thinking is expected to benefit software development. The basic idea in this applying is similar to the original use of Lean thinking: an attempt is made in order to produce only the right amount or number of the product, and only then when someone needs the product, which should eliminate Muda, Mura, and Muri. As a result, less waste should lead to more value.

The literature above prososed that original Lean practices cannot be adapted straightforwardly into software development. Meanwhile, the literature suggests that the Lean principles can be used to derive such practices that take into account the needs of the particular nature of software development (regarding waste elimination and JIT, for instance).

Several authors are suggesting that Lean provides more possibilities to optimize software development projects appropriately but contains a risk to "eliminate" too much waste, in contrast to the agile approach. Nevertheless, Kanban's demands to visualize the progress, limit WIP, and measure the cycle-time are, according to the literature addressed above, supposed to be an answer – at least to some extent – to issues of the agile software development (presented in chapters 1 and 2.4). The small number of empirical evidence, however, needs more research in order to validate what the real impacts of Kanban are for software development.

Chapter 3

Determining Software Project Success in Lean Thinking

In order to link Lean thinking with factors affecting project outcomes, Chapter 3.1 defines the concept of project success while Chapter 3.2 introduces a general framework for project success and maps the relationship between the framework and Lean software development.

3.1 Project Success

Regarding project success in software engineering, the literature still reports serious failures – despite the half-century history of software projects. World-widely, less than half of the projects are considered successful, argued anecdotally by Ernest-Jones (2007); Hartmann (2004). Nevertheless, criticism against the validity of these arguments has been presented by Glass (2006), for instance.

The term "success" means different things to different people (Freeman and Beale, 1992). Correspondingly, "project success" is typically understood differently between the internal and external organizations of projects (developers compared with customers, for example). Understandings collide even inside organizations (Agarwal and Rathod, 2006). Awareness of this ambiguous situation offers a new viewpoint to consider the validity of the reports regarding the success of software projects: such reports only expose that there is something wrong with the projects from a certain point of view. Whether this "wrong" is relevant depends on the project context and the viewpoint. For example, one organization may consider its project to be successful even if the schedule has been exceeded seriously. Meanwhile, another organization may think such a delayed project is an

economical disaster despite the satisfaction of the customer.

The definition of PMI (2008) for project success consists of product and project quality, timeliness, compliance with the budget, and customer satisfaction. Meanwhile, Shenhar et al. (2001) divide project success into four dimensions based on empirical evidence: (1) project efficiency (meeting time and budget goals), (2) impact on the customer (such as customer satisfaction, meeting operational and technical specifications, and solving a major operational problem), (3) business success (commercial success and a market share), and (4) preparing for the future (opened a new market or line of products and developed a new technology). This division looks similar to the definition of the PMI.

As one key for a better performance level of the software engineering, software tools have been considered. As early as the 1980s, programming productivity and quality were found to be improved by using these tools (Boehm, 1981; Thadani, 1984; Jones, 1986). Even though the right kind of automation is desired and thereby expected (Ravichandran and Rai, 1994), understanding the knowledge, tools, and techniques is insufficient for effective project management (PMI, 2008, 13). Automatition, indeed, in software development support has been over-emphasized since the middle of the 1980s (Conradi and Fuggetta, 2002). Because software development is a largely dynamic and cooperative set of activities, methods, and transformations used by people, the impact of software development technology with its modeling languages, editors and interpreters has only been small (Conradi and Fuggetta, 2002). In those days, crucial processes, such as learning, technical communication, requirements negotiation, and customer interaction, were poorly described in software process models (Curtis et al., 1988).

Being aware of the existence of problems in software development that may lead to project failure is insufficient for being aware of the reasons for such problems. Thereby, the following sections take a deeper look at the problems and focus on their reasons.

3.1.1 Individual and Team Levels

In order to improve software development, the overall development should be focused on (Petersen, 2010, 40). By doing so the success of a project may not, however, be guaranteed. Without understanding human relations, project managers may have difficulties in controlling their customers or even their own software teams. The first problems may occur right at the beginning of projects. According to Hutchings et al. (1993), people after their group begins to meet typically focus on belonging in the group and

on orienting themselves: they sit back waiting and collecting more data before really joining the group. Team-based learning, group behavior, and ways in which people create purposes and communicate are key elements of the group wherein they form (Hutchings et al., 1993). The five-stage theory of Tuckman and Jensen (1977) about growth and development of the groups emphasizes the meaning of people. It states that groups need changes to grow. According to the theory, the team development starts by "forming" the group and ends in the "adjourning" via the "storming", "norming", and "performing" stages. Maturing requires time and after groups grow, they become more efficient (Tuckman and Jensen, 1977). One reason why exceptional results and the highest levels of performance remain unreachable in some groups, is that they never complete the storming stage (Pugh, 1991; Schein, 1988).

Boddy and Macbeth (2000) discover four practices explaining successful implementation of projects: goals, resources, structures and controls. These practices create the need for managers to focus on the following five areas: (1) ensuring agreement with goals, (2) obtaining resources, (3) monitoring and learning, (4) exercising influence (using individual initiative and creating appropriate structures), and (5) ensuring effective communication (Boddy, 2002). In this sense, many modern projects have no tangible outputs (Maylor, 2001). The result of a software engineering project is no more than a shared database with an integrated system if the time constraints, tasks, milestones, and cost are the only concerns for the project manager, actors and stakeholders (Gunson et al., 2003). The ability to allocate both technical and human resources has been identified as one of the key factors in commercial software development over three decades ago (see Zmud (1980)).

Regarding the human aspect, the PERFORM model states characteristics for a high-performing team. The acronym is based on the following (Blanchard et al., 2004, 12–13):

- (P)urposes and values remind of the team's clear commitment to a common purpose. Common values improve integrity, quality and collaboration.

- (E)mpowerment means availability of relevant organization and business information, as well as initiative, involvement and creativity encouraged by values, norms and policies.

- (R)elationships and communication stress the meaning of respect toward differences in the sense of ideas, opinions, feelings, perspectives,

and cultures. Attentiveness includes honest and caring feedback and understanding.

- (F)lexibility consists of shared responsibility, using unique talents and strengths, and openness to explore different ways of working. Calculated risks are also supported.

- (O)ptimal performance constantly builds up production of significant results. The team commitment is required for high standards and measures for productivity and quality. The team learns from mistakes and improves continuously.

- (R)ecognition and appreciation happens, when team contributions are recognized and valued by the larger organization. A feeling of high regard within the team is common and accomplishments are acknowledged.

- (M)orale is based on confidence and enthusiasm about the team's efforts. The sense of pride and satisfaction is strong and the members help each other.

Meanwhile, singular issues considered as critical success factors have been reported in the literature. Over two decades ago, Bullen and Rockart (1986) concluded things that must go right for a successful project. These things include clearly defined objectives, adequate budget, realistic schedule, customer or user participation, project leadership, change control and management, communications, and problem solving. Curtis et al. (1988) and Jurison (1999) agree: the most salient problems reported in projects concerning additional efforts or mistakes include communication and coordination breakdowns.

Lack of shared understanding in and between organizations shows as hiding of the decision-making process and as decision-making without authority approval (Jensen and Scacchi, 2005). Teasley et al. (2000) even show that the productivity in teams can be doubled by easing their access to each other and by making work artifacts visible to all. Customer perceptions of quality have a direct influence on information system providers and their products (Licker, 1992), which emphasizes the meaning of cooperation between producer and customer: producing even the best quality from the viewpoint of a producer may not satisfy a customer if the conceptions of the producer and customer regarding quality differ from each other.

Team members have to develop certain concepts as a common approach to getting things solved and done. Successful teams do not accept the

obvious causes but think their way around the problem thoroughly (Robson, 1993; Poppendieck and Poppendieck, 2003).

3.1.2 Organization Level

Executives in every domain of business know that their ability to compete directly depends on how they can organize people and motivate, develop, attract, and retain talented people (Curtis et al., 2001). In addition to individuals, success has been shown to depend on organizational factors and experience, as well: Capability Maturity Model Integration (CMMI) focuses on the following issues: (1) characterizing the maturity of workforce practices, (2) establishing a continuous program to develop the workforce, (3) setting priorities to form improvement actions, (4) combining workforce development and process improvement, and (5) establishing a culture of excellence (CMMI Product Team, 2007). However, Middleton and Joyce (2011) criticize this based on their literature review: "After 20 years in existence, the independent evidence that CMMI leads to improvements in product cost, quality, and timeliness is slowly accumulating." Besides, more practices than those in CMMI are needed to motivate, develop, attract, and retain top software talent (Curtis, 1994). As a solution, the People Capability Maturity Model (People CMM) (Curtis et al., 2001) is based on human-related issues in software projects and is supposed to guide management and development of the workforce, and to help address critical people issues. It establishes successive foundations for (1) continuously improving individual competence, (2) developing effective teams, (3) motivating improved performance, and (4) shaping the workforce that an organization needs to accomplish future business plans (Curtis et al., 2001).

Many project managers use involvement as a risk control method for management, users, and the steering committee (Addison and Vallabh, 2002). Management involvement especially decreases the risks of unclear or misunderstood scopes or objectives, unrealistic budgets or schedules, and continuous requirement changes (Addison and Vallabh, 2002). Even 11 years before this identification, Boehm (1991) ranked the two latter factors among the top six of the software risk factors. Moreover, other risks occurring in software projects regularly are lack of senior management commitment to the project, failure to gain user involvement (i.e., developers may have to make assumptions about functionality details and objectives when users are not involved), inadequate knowledge or skills (e.g., knowledge of the personnel in technology, business, or project handling), developing the wrong software functions (from the user or technical viewpoint), subcontracting (e.g., shortfalls in externally developed components), resource

usage and performance, introduction of new technology (not successfully in use in other organizations thereby being risky), and failures in managing end user expectations (Addison and Vallabh, 2002). Most of these risks are recognized as culture-independent (Keil et al., 1998).

According to Weiss et al. (2002), in as early a process as a software assessment, the first step should be to obtain commitment from the managers of the organization: when senior managers believe that the process will benefit them, the level of commitment of participants begins to increase. On the other hand, Abrahamsson (2000) stresses a necessity for management commitment and concludes that this commitment is useful only in order to enable the process improvement culture to grow.

3.2 Framework for Project Success

Chapter 3.1 briefly described the fragmented status of the area of project success in software development. Due to this fragmentation, Chapter 3.2 focuses on targets of improvements for practitioners' software work. This is done with the aid of the Lean approach because, based on the literature addressed in Chapter 2.3.3, (1) Lean thinking appears to be possible to transform to software development and (2) the potential of Lean thinking to software development was found to be promising. Therefore, we present our framework, part of our study contribution (Paper II), for project success (Figure 3.1). This general framework presents themes and constructs that have been recognized to have influence on software development projects. In the figure, they have been divided into levels: individual and team, project, and organization. Later, Figure 5.1 will illustrate the role of the themes and constructs regarding the research in this thesis.

The themes and constructs (i.e., the white boxes in Figure 3.1) are a contribution of our studies (Paper I and II) and have been derived from the literature-based theories, models, methods, tactics, and best practices. Based on the empirical evaluation, driving or restraining factors of projects can be traced to these themes and constructs. The division into three forces (i.e., organizational (decision power), internal (stress), and external (dynamics)) has been adopted from the work of Bertelsen and Koskela (2003). By following that division, the themes and constructs have been placed in the forces according to their primary function to best fit the description by Bertelsen and Koskela (2003) regarding the contexts of the forces. The model of Bertelsen and Koskela (2003) has been designed for handling chaos[1] in construction projects with the aid of the three forces.

[1]Bertelsen and Koskela (2003) explain factors affecting project success with chaos:

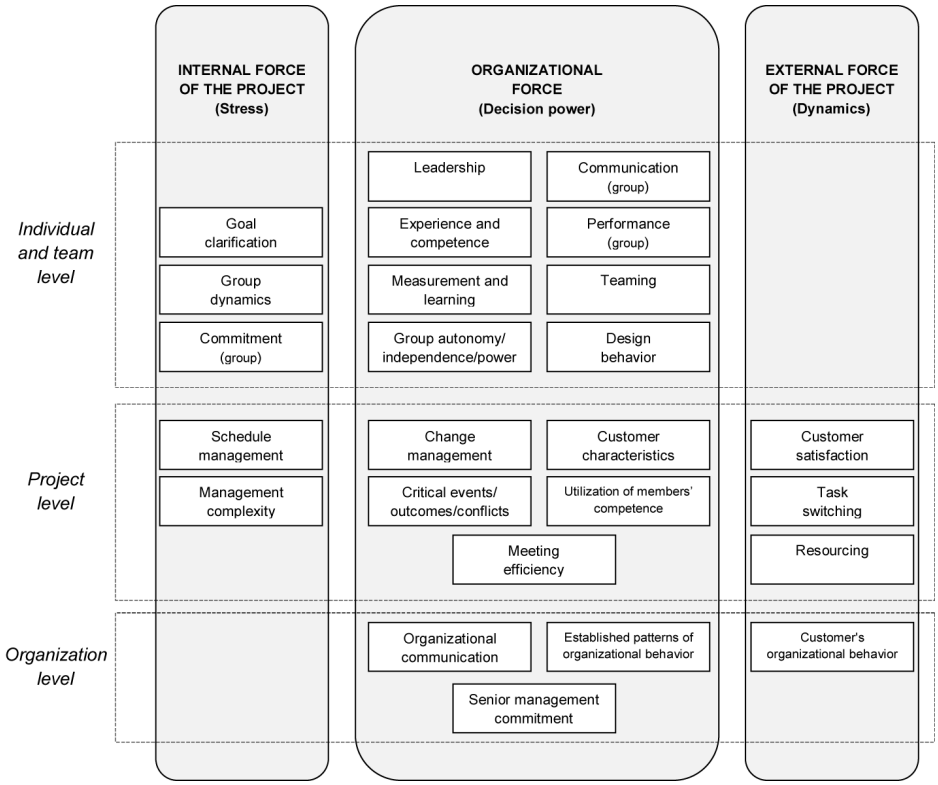

Figure 3.1: A general framework of software engineering project success with the three forces and their themes and constructs (Paper II).

Nevertheless, the model is assumed to be suitable for software development projects, as well, due to similar theories of project and management (Koskela and Howell, 2002). Thereby, the model is used as the foundation for our framework.

In general, Law et al. (1998) divide relationships between the dimension (here a force) and the construct (here a theme or a construct) into three: latent model, aggregate model, and profile model. According to this taxonomy, our framework is determined as the profile model. This means that the forces of the framework are not supposed to combine because of their theoretical nature. Rather, the profile model collates the themes and constructs as various combinations of their forces' characteristics. There are no weights between the themes and constructs (in contrast to the ag-

chaos may be channeled into increased cost and duration and into reduced value and functionality of a project.

gregate model). Neither can any formula determine the level of success in our framework because the framework has not been formed as an algebraic composite of its forces. Likewise, the framework is not appropriate to interpret as the latent model due to its hierarchy: the first-order factors do not equally manifest success. For these reasons, the framework follows the characteristics of the profile model.

The following three sections focus on the content of each force of the framework from the viewpoint of Kanban and Lean thinking.

3.2.1 Decision power

The organizational force (Figure 3.1) related to the individual and team level appears to contain fundamental parts of Lean thinking. This is showed by walking through the themes and constructs as follows.

Shared **leadership** and **communication** have been highlighted in Lean thinking where it is not considered reasonable to keep all the decision power in the hand of project managers (Chapter 2). The principle of empowering the team (Chapter 2.3.2) allows projects to reach an efficient decision-making policy: where the knowledge is, there the decision power is. In a team of high **performance**, this means that everyone can take responsibility, which, on its part, requires communication ability (Poppendieck and Poppendieck, 2003). This ability is one key for successful use of Kanban as mentioned above and as will be argued in Chapter 5.

In addition to communication, **experience and competence** are needed in projects. Decisions and solutions will be fact-based rather than random guesses if relevant experience and competence are present during the decision-making. This is true also regarding re-tailoring and optimizing Kanban for different project contexts: once the needs and limits in a certain environment are known, re-tailoring can be made based on facts, not guesses. According to the principle of deferring commitment (Chapter 2.3.2), a lack of needed information results in either stopping the work to find the answer or guessing without stopping (Chapter 2.3.2).

"Amplify **learning**" (Chapter 2.3.2) is one of the suggested principles for Lean software development. The progress of a flowing project should be **measured** in order to detect problems (Reinertsen, 2009). This is expected to lead to Kaizen. Short feedback loops accelerate learning since mistakes are revealed faster and doing mistakes can be stopped earlier (Takeuchi and Nonaka, 1986; Mandić et al., 2010). Kanban enables this learning by allowing changes and feedback and by limiting overprocessing (by setting the WIP limits) (Chapter 2.5). This executes Heijunka, Hoshin-kanri, and Jidoka.

Teaming is part of a self-organized, high-performing team (Cohen and Bailey, 1997; Guzzo and Dickson, 1996; Moe et al., 2009a). Software development work is carried out mostly by teams, not by individuals (Sommerville, 2007). A positive atmosphere of trust, morale, and motivation disappears if a group puts its energy into fighting each other instead of into working (Blanchard, 2001). **Group autonomy**, as already stated, is part of a self-organized Lean team.

Design behavior includes brainstorming, clearing up encountered problems, looking for more than one basis and solution (Yokoten), finding necessary information to solve a problem, and modeling possible solutions and implementing experimental solutions before making a decision. Lean principles (Chapter 2.3.2) support this kind of design behavior: "respecting people" values people's skills and knowledge and encourages projects to utilize them. Necessary knowledge is created ("create knowledge") for the right decision, and the commitment is allowed to be deferred ("defer commitment") in order to find this right decision. Making the right decisions, on its part, helps to "build quality in" and "eliminate waste". A diminished number of obstacles and restraining elements helps the team to follow the Lean principle of "deliver fast," as well.

Conventional software development approaches the **change management** in a different way than Lean practices. Lean practices take into account the fact that the content of complex, abstract-term software development projects cannot be known at the beginning (Chapter 2). Therefore, when a change occurs, teams using Lean practices can adapt to the new situation in a more flexible way than using the conventional practices (Poppendieck and Poppendieck, 2003). Kanban notices this dynamic situation of modern projects (Gross and McInnis, 2003). As a flexible method it allows overriding the planned implementation order of the tickets: a new, important ticket can be started to carry out as soon as someone gets the previous ticket carried out and is thereby free to engage in this new one (Kniberg, 2009).

Committing the customer to the project requires interaction. The cooperation between the producer and customer should be fluent despite potential unwanted **characteristics of the customer**. Even though agile practices emphasize customer interaction more than Lean practices (Chapter 2.4), adding customer-value is possible when the team is aware of the customer's needs (Chapter 2.3). Implementing successfully the Lean principles of "creating knowledge" and "optimizing the whole" require cooperation. The pull mechanism used in Kanban strongly supports this cooperation since the customer can "pull" the most relevant, the most value-adding

features first by prioritizing them appropriately (Chapter 2.5).

Critical events can be seen from both the general and project-specific viewpoint. Many problems related to the plan-driven approach, such as lack of feedback, can be an issue in the Lean context, as well. Lean principles, however, encourage people to give feedback, which executes Kaizen (Chapter 2.3.3).

Ability to **utilize members' competence** should increase the team's competence. While redundancy is useful, there may not be time enough to educate everyone in the team into specialists in each relevant area. Rather, different skill areas of the members should be utilized. The question is about "respecting people" and "creating the knowledge" (Chapter 2.3.2).

The Lean principle of "eliminate waste" concretely orders us to eliminate waste from the projects (Chapter 2.3.2). This includes keeping the **meetings effective**. Although meetings are mandatory in projects, discussing irrelevant issues in the meetings, for example, does not add any customer value. Kanban provides visual information about the progress and helps people to "optimize the whole" (formerly "see the whole"), which also helps members to make fact-based decisions in the meetings (Chapter 2).

The principle of "optimize (or see) the whole" (Chapter 2.3.2) provides for communication also beyond project boundaries (i.e., **organizational communication**). In other words, communication is not a requirement for project teams only. **Patterns of organizational behavior** support this principle: business processes, technology, resources, structure, people, culture, and the policy of decision power of the organization should be aligned with the project. In this way, the project gets support from the organization. This support also includes the **commitment of senior management**. The Kanban board provides important information to the management level, too (Chapter 2.5).

Table 3.1 summarizes the mapping presented above.

3.2.2 Stress

Internal force (Figure 3.1) deals with issues occurring inside projects. When a project has clear goals and its members have a shared direction, attempts can be made to attain this direction and the whole can be seen ("optimize (or see) the whole"). Mandić et al. (2010) connects uncertainty to waste and thereby suggest to minimize it. Regarding **goal clarification**, Lean thinking concretely emphasizes adding value to the customer (Chapter 2.2.2). Kanban's visuality is expected to prevent misunderstandings and keeping the cycle-times short allows fast feedback (Chapter 2.5).

Themes/constructs	Mapping	References
Leadership	Empower the team	Chapter 2
Communication	Empower the team; respect people	Chapter 2
Performance	Empower the team	Poppendieck and Poppendieck (2003)
Experience and competence	Defer commitment	Chapter 2.3.2
Measurement and learning	Amplify learning; Kaizen; feedback	Chapter 2.3.2 and 2.5, Reinertsen (2009); Takeuchi and Nonaka (1986); Mandić et al. (2010)
Teaming	Self-organization	Cohen and Bailey (1997); Guzzo and Dickson (1996); Moe et al. (2009a)
Group autonomy /interdependence /power	Self-organization	Cohen and Bailey (1997); Guzzo and Dickson (1996); Moe et al. (2009a)
Design behavior	Yokoten; respect people; create knowledge; build quality in; eliminate waste; deliver fast	Chapter 2.3.2
Change management	Adapting to the changed situation	Chapter 2, Poppendieck and Poppendieck (2003); Gross and McInnis (2003); Kniberg (2009)
Customer characteristics	Interaction with the customer	Chapter 2.3, 2.4, and 2.5
Critical events/ outcomes/conflicts	Kaizen; feedback	Chapter 2.3.3
Utilization of members' competence	Respect people; create knowledge	Chapter 2.3.2
Meetings efficiency	Eliminate waste; optimize the whole	Chapter 2
Organizational communication	Optimize the whole	Chapter 2.3.2
Established patterns of organization behavior	Optimize the whole; respect people	Chapter 2.3.2 and 2.5
Senior management commitment	Optimize the whole; respect people	Chapter 2.3.2

Table 3.1: Mapping the organizational force with Lean thinking.

Themes/constructs	Mapping	References
Goal clarification	Value-adding activity	Chapter 2.2.2 and 2.5, Mandić et al. (2010)
Group dynamics	Respect people	Chapter 2.3.2
Commitment	Hansei	Chapter 2.2.1
Schedule management	Flexibility; changing the plans	Chapter 2.5, Poppendieck and Poppendieck (2003)
Management complexity	Flexibility; changing the plans	Chapter 2.5, Poppendieck and Poppendieck (2003)

Table 3.2: Mapping the internal force with Lean thinking.

Group dynamics is also a part of Lean thinking. The principle of "respect people" (Chapter 2.3.2) concerns, among other things, trust and an open atmosphere within the project. Such an atmosphere encourages members to ask for advice when needed, even though it reveals the asker's incapability in that issue. The **commitment** of the team is a part of a successful software development project in both conventional and Lean context (Hansei). Successful use of Kanban provides for this commitment, too (Chapter 2.2.1).

While **scheduling management** and **management complexity** are approached mechanically in conventional software development, Lean practices of software development do not believe in foreseeing schedules accurately or taking into account all problems beforehand (Poppendieck and Poppendieck, 2003). Instead, Kanban allows flexible circumstances for reprioritizing and changing the plans (Chapter 2.5).

Table 3.2 summarizes the mapping presented above.

3.2.3 Dynamics

While decision power and stress may be controlled inside a project, some issues coming from outside may not. Projects interact with the external force (Figure 3.1) without power to control it (Bertelsen and Koskela, 2003). **Customer satisfaction** belongs to this force and is a part of Lean thinking, namely, value should be added to the customer as addressed (Chapter 2.2.2). Kanban helps the customer to realize that he cannot have everything at the same time, which makes him really think about the most important thing that the project should carry out first (Chapter 2.5).

Themes/constructs	Mapping	References
Customer satisfaction	Value-adding activity; task prioritizing	Chapter 2.2.2 and 2.5
Task switching	Muda	Chapter 2.2.2 and 2.5
Resourcing	Optimize the whole	Chapter 2.3.2
Customer's organizational behavior	Create knowledge; see the whole	Chapter 2.3.2

Table 3.3: Mapping the external force with Lean thinking.

Due to considering **task switching** to be waste (Muda) in Lean thinking (Chapter 2.2.2), organizations or projects should not overload people by assigning them to several tasks at a time. Individuals, in their turn, should take care that switching between their tasks is minimized or eliminated whenever possible. Kanban attempts to prevent over-processing and task switching by requiring WIP limits (Chapter 2.5).

Regarding the resources mentioned above, **resourcing** is part of the Lean principle of "optimizing the whole" (Chapter 2.3.2). After the organization has provided adequate and appropriate resources for the project, it is up to the project to utilize them in the best way.

Finally, even though a project is not able to manage the **customer's organizational behavior**, the cooperation with the customer provides opportunities to reveal misunderstandings and to recheck the meaningfulness of the customer's needs ("create knowledge" and "see the whole"): in this way, true value may be easier to produce by following Lean thinking.

Table 3.3 summarizes the mapping presented above.

3.3 Summary

Chapter 3 showed the multiple nature of project success and software project management. Moreover, it suggested a framework for project success that was then analyzed from the viewpoint of Chapter 2. As a result, the project success framework introduced critical areas that the literature has shown to have a connection with project failures or performance. Poppendieck and Poppendieck (2003) state that Lean practices appear to have the ability to answer challenges in these kinds of areas. Chapter 3.2 conducted one scenario of how to map these challenges with the aid of Lean thinking.

Like the multiple nature of the projects, also the reasons for failures were

recognized to have a multiple nature. The software products are unique to other products: they are invisible, which is one main reason for the difficulty of software development; they are complex, actually the most complex constructs man has ever created; and they are changed throughout their entire lifetime because changing them to better fit changed environments is easier than vice versa (Brooks, 1987). Thereby, the findings related to various areas of problems are not surprising. The value here comes from the strategy of tackling conventional challenges of software development with the aid of the Lean approach. Having addressed targets of improvements in the area of software developing and introduced how Lean thinking could be applied to it, a need to conduct studies regarding the area arises.

Chapter 4

Empirical Research Design

This chapter describes the research approach and methods adopted for the research papers of this thesis. Chapter 4.1 briefly introduces common research strategies and methods for software engineering. Chapter 4.2 presents the research approach and Chapter 4.3 research methods applied in the research of the thesis. The latter chapter focuses on the chosen research methods from the viewpoint of the study in the thesis. Finally, Chapter 4.4 describes the collection of the empirical evidence while Chapter 4.5 introduces the research context of the case projects.

4.1 Studying Software Engineering

At a high-level taxonomy of research approaches, two main types of empirical studies can be categorized into qualitative and quantitative. Qualitative research methods have been developed mostly in social disciplines for focusing on humans and their actions and relations. The qualitative research type refers to studies wherein data is gathered from humans directly or through their actions. The techniques include interviews, surveys, and observations. As a result, data gathered in such a way is to some extent subjective and biased because beliefs and personal standards may affect the viewpoints of the subject or observer. Qualitative data cannot usually be analyzed mechanistically. It needs, instead, to be interpreted in some way, which risks a bias (Seaman, 1999; Wohlin et al., 2000).

The quantitative research type, in contrast, originates from the natural sciences wherein data is available through measurement. This can be done by using instruments and calculations or by direct empirical observations. The type quantifies properties of the phenomenon being studied and it produces data as numbers. Doing sample-based statistical calculations

requires plenty of data (Wohlin et al., 2000).

Regarding empirical strategies, Wohlin et al. (2000) categorize them into three high-level types as follows.

- Survey: A retrospective type of study. Data can be quantitative, qualitative, or both. Data are collected after an event, typically by questionnaires or interviews. Interpreting data produces either explanatory or descriptive conclusions.

- Case study: An observational type of study focusing on a phenomenon over time. Both qualitative and quantitative data can be gathered alone or in combination. This combination, i.e., triangulation, has the benefit that one type of data may complement or support another type.

- Experimentation: In contrast to a survey, a high-level-of-control type of study wherein controlling the variables of the subject being studied is possible. Here, control means being aware of the values of the variables through some kind of manipulation. Moreover, the values can be measured with a certain known level of accuracy. Three types of variables are as follows.

  - Dependent variables, i.e., response variables, represent the output or effect in the experiment, such as productivity or quality of a development project.
  - Independent variables are controlled in the experiment setting, such as different leadership style between different experiment groups. As a result, the experiment figures out how the dependent variables are influenced by the independent variables or how the changes in variables correlate.
  - Moreover, context variables affect the dependent variables, which emphasize being aware of them in order to evaluate the relationship between the dependent and independent variables.

Variations and other types exist since the field of software engineering still lacks a commonly accepted taxonomy of empirical strategies (Wohlin et al., 2000). Järvinen and Järvinen (1996), for example, divide the research process into the following five categories.

- Theory-testing research. The idea is that the hypotheses have been derived from a theory and we like to test their reliability. This research category includes controlled experiments, field methods (such

as field study, field experiment, and survey), theory-testing case studies (including intensive, comparison-focused, and action research), and theory-testing longitudinal study.

- Descriptive, interpretive, and theory-creating research includes grounded theory (i.e., the method suitable for creating a theory that is based on research data), case study, phenomenography (that illustrates how people conceptually parse the world), contextualism (suitable for change processes and in more general for longitudinal studies), and ethnographic method (wherein a researcher visits the study object (e.g., an organization) for a longer time and is carefully orientated into the functions of the object).

- Constructive research is beyond basic research and answers the question, what kind of a world it is. Typically, this research category attempts to create a new reality that is based on existing (research) data or information.

- Mathematical research focuses on adequate laws of the variety and hierarchy, classification of causal systems, and a-posterioric knowledge formation.

4.2 Research Approach

The research approach in the study of this thesis was systematically organized into two parts. The first part analyzed software development project success. Moreover, this part explored self-organization in order to set sample criteria in the second part. After having showed why projects fail, the second part explored Lean software development. Based on the literature addressed in Chapter 2 and 3, an assumption was made that Lean thinking is able to benefit software development. For this reason, the second part conducted a series of case studies in order to explore impacts of Kanban on software development projects. In addition, a quasi-controlled experiment strategy was used to aid in understanding the whole better than could have been understood with only a single research strategy. The qualitative research type was selected for the study due to the explorative nature of the phenomenon being studied. Table 4.1 presents which research question is considered in which research paper, and the empirical research strategies used.

Research Question	Paper	Research Strategy	Number of Case Projects	Focus
RQ1.1	I	Case Study	8	Conceptualizing project success of software development projects.
RQ1.1	II	Case Study	1	
RQ1.2	III	Case Study	2	Self-organizing teams in Lean-based Kanban-driven software development.
RQ1.3	IV	Case Study	1	Exploring whether the Kanban process model can reveal waste in software development projects.
RQ1.3	V	Case Study	1	Impacts of Kanban on software development project work.
RQ1.3	VI	Quasi-controlled Experiment	2	The role of management in Lean-based Kanban-driven projects of self-organizing teams.
Total	**6**		**15**	

Table 4.1: The research questions, papers and methods used in this thesis.

The first research question focused intentionally on the conventional software development process. This was because the conventional approach has been studied widely and because of its historically strong and long-term impact on software development projects. Since such corresponding research would yet have been inappropriate in the quite novel and unexplored context of Lean software development, the first two research papers (Paper I and II) together formulated a foundation for exploring the Lean context: these papers showed why plan-driven software development projects may fail even with professional, experienced teams. While achievements of Lean manufacturing have been widely reported in the literature (Chapter 2), the question arises, what the impacts of Lean-based Kanban are on software development.

Each study was qualitative, although the quantitative manner and triangulation (Chapter 4.1) were also applied to them. The data collection consists of multiple evidences including documentation, interviews, direct and video observation, and participant observation, as suggested by Yin (1991, 85–94). The executed interviews followed the variation of the semi-structured theme interview (Patton, 1990). This approach of the multiple evidence enables a more objective view for the analysis than a single-type evidence (e.g., interview) only (Parnas and Curtis, 2009).

According to the dialogical reasoning principle in interpretive research, an improved understanding of the previous research stage should become the prejudice for the next research phase (Klein and Myers, 1999). Each paper follows this principle by containing more than a single research phase only.

4.3 Research Methods

This chapter describes the reasons for selecting the research methods used in this thesis. Moreover, it explains how the data collection and analysis were applied in the study.

The intention behind the study was investigating contemporary phenomena of a real-life context: what influences does Kanban have on Lean software development projects. Case study was chosen as the main research method for the study. This was because it enables collecting data through observation in an unmodified setting and allows capturing of details and the analysis of many variables (Yin, 1991, 13–14). The four steps of a case study method are (1) preparing data collection, (2) collecting evidence, (3) analyzing the evidence, and (4) reporting the studies (Yin, 1991).

The case study method, however, has been criticized for a lack of gener-

alizability (Yin, 1991, 21). Typically, the context of each case study differs from each other (Kitchenham et al., 2002). Walsham (1995) reminds us that case study is suited, instead of generalization, for looking for plausibility and logical reasoning through developing concepts, drawing specific implications, and contributing rich insight into the phenomenon under study. Regardless, this was not considered problematic between the studies conducted in the thesis because the goal was to explore influences of Kanban rather holistically than from a single point of view. Moreover, the thesis rather focuses on individual than general phenomena in real life since, as stated in Chapter 2: Lean software development seems to be too immature a field of research to generalize yet as in its entirety.

In addition to the case study research method, the quasi-controlled experiment method (Wohlin et al., 2000) was also used since *Software Factory* (Chapter 4.5.2) made its use possible. This research method was inevitable when two projects were compared with each other (Paper VI). The method is a variation of the experimentation introduced in Chapter 4.1. The main difference is the number of context variables.

Research in this thesis focused on individual and team levels. The third level of the division of three by Hovorka and Larsen (2006) is organization. A team cannot exist without individuals. The same is valid with organization, hence limiting studies purely to one level only would have been inappropriate. In software development projects, the work is usually done in groups. Customers, representing the organization level here, were used as evaluators of the case projects under the study of the thesis to determine project successes and to give viewpoints that differ from those of the producer's project team members. The level of the individual was chosen as the main focus for the thesis because it is a base for understanding the team level. The project success framework (Figure 3.1) supports this line of thinking.

4.4 Collection of Empirical Evidence

The personal face-to-face interview is considered as an efficient data-gathering technique particularly for interpretive studies (Yin, 1991, 19–20) wherein our research is also categorized. In addition, information gathered is likely to be more accurate than information collected by other methods (Oppenheim, 1992). An interviewer can, for example, explain the questions in an understandable way in order to avoid incomplete answers or misunderstandings. In laboratory conditions, video observation does not disturb the subjects being studied. Direct observation (i.e., an interviewer observes

Study (Paper)	**1**	**2**	**3**	**4**	**5**	**6**	*Total*
Individual interviews	40	7	5	4	10	6	**72**
Direct observation			yes	yes	yes	yes	**yes**
Participant-observation			yes				**yes**
Video observation				yes	yes	yes	**yes**
Questionnaires	120	28		24	24	18	**214**

Table 4.2: Data collection of the studies in the thesis.

in the project space) enables sensing the mood and feelings of the group or individuals under observation. By investigating documentation, such as the project manager's diary or production system, helps to perceive things that are not present on video. Participant observation was possible only in one case.

The data collection was gathered during the period from 2007 to 2010 covering totally 15 projects and 72 individual interviews as presented in tables 4.1 and 4.2. In addition, feedback was gathered from the customers regarding the projects in order to evaluate the projects' success. All the interviews were recorded in audio and transcribed using a systematic codification system. The duration of the interviews in study 1 were 40 minutes on average while in study 2 they were 45 minutes on average. For the rest of the studies, the interviews (the Kanban cases) varied between 60 and 120 minutes. In the pilot study (study 1) each participant in the project groups including the project managers were interviewed. In the other studies, the population samples were selected to represent members comprehensively in each group and both seniors and juniors. This is called a role-based sampling. The project managers were interviewed in each project of every study.

4.5 Research Context

The case projects performed in three different contexts that are described in the following.

4.5.1 Capstone Software Engineering Project Environment

This setting was used in Paper I as a pilot case study (Yin, 1991, 80–82) for the study in Paper II.

The Software Engineering Project is a mandatory course at the Department of Computer Science, University of Helsinki. Students take the course after they have passed most of the Bachelor-level courses, typically, in their third year. Before the course, the students have learnt software

requirements, design, construction, and evolution: these constitute Core Area Components, one part of *Guidelines for Software Education* (Hilburn et al., 1998). In addition, the students know some details of architecture, are capable of adhering to standards and practices, and have experience of working in small groups from earlier courses. These issues have been proposed as a set of requirements for educational processes in software development (Filho, 2001). The project imitates a real-world software project although the students cannot influence the group composition, the duration of the project, or the resources allocated to the project. The grading of the course takes into account both individual and team performance to increase motivation, as proposed by Hazzan (2003). The supervisor of all the capstone projects in the course furnishes outside customers and ensures that the grades between the project groups are aligned for every group and group member. Grades for the groups are based on suggestions from the instructors and customers. Instructors typically are involved with at most two project groups, and customers with just one. The course lasts for 14 weeks. Students are (based on the interests they have expressed, but in practice, fairly randomly) grouped into groups. The group size is typically 5 to 6 students, no less than 4. Each group has to develop a software product that their customer orders at the beginning of the project. The customer may be from the Department as well as an outside partner or a commercial company. Every group is assigned an instructor (Faculty member of the Department) who monitors the group throughout the project but helps the group only on a general level. For example, all support in planning or implementation is excluded. However, the instructor may give practical advice, for example, on project management or role or task coordination.

The groups choose the project manager among the group members, as well as other roles, such as the implementation or document manager. Project managers are usually inexperienced in the sense that they have not been managers in any real-world projects, even though some of the students do have genuine work experience in paid programming projects.

4.5.2 Experimental R&D Laboratory 'Software Factory'

This setting was used in Paper III, IV, V, and VI.

Software Factory[1] is a new software engineering research and educa-

[1]Not to be confused with the early "software factories" in Japan or in the USA of the 1960s to 1980s. The first software factories focused on making software production in manufacturing-like systematic ways with CASE tools and component reuse (Cusumano, 1991). Some modern software engineering application frameworks have also coined the

tion setting at the University of Helsinki (Abrahamsson, 2010). It is basically an industry-oriented R&D laboratory environment for conducting software business projects. This executes an Experimental and Explorative Research (EER) strategy that stems from the needs of explorative research and empirical research (Oivo et al., 2004). Even though EER is industry-oriented, Software Factory utilizes the idea. The concept of Software Factory comprises a physical laboratory environment coupled with a novel operational model. The laboratory room is equipped with sophisticated computer and monitoring equipment and equipment for software development (e.g., Smartboards). Such high-end facilities make it possible not only to conduct actual software engineering work in a modern fashion, but also to collect online research data automatically (e.g., logs). Moreover, the facility enables capturing rich insights into the human-related aspects of software development (Fagerholm, 2010). The entrepreneurial aim of Software Factory is to conduct business-driven software development projects for creating new product prototypes and commercializing them (possibly with spin-offs). All this can be utilized as a research platform and case study environment. The Department of Computer Science hosts an initial reference implementation, with more locations coming up globally.

The software development teams are required to be self-organized and they are responsible straight to their customer. The teams consist of experienced Master-level students who have been selected based on qualification. There is no external instructor or coordinator for the projects. A technical consultant and a coach, however, are available for the teams without charge.

4.5.3 Real-life Business

This setting was used in Paper II.

The case study covered a software developing project in a global, multinational software development organization. The case project evaluated was part of a bigger program which followed the waterfall process model (Royce, 1970) trends but in an iterative way. A preliminary requirement analysis project preceded the case project. Once this analysis project finished, the customer needs had already changed. The actual case project contained iterations.

The case organization attempts to resource projects so that the participants are assigned to one project at a time. Nevertheless, some of the members of the study object also had other duties. The organization is

term "factory" (Greenfield et al., 2004). However, such facets can be hosted in the Software Factory environment, as well.

experienced in producing software systems for its customers. People are respected by the organization and their well-being is considered high in both employees' and their managers' point of view. Motivation of the workers has been reported to be higher than average on the field. People are assigned to projects based not only on their skills but also on their interests. The multinational synergy advantages are utilized and information and knowledge are shared, not hided, between projects.

Chapter 5

Research Contribution

As stated in Chapter 4, the research papers of this thesis focused on two parts. In the first part, Paper I served as a pilot study for the study conducted in Paper II. While the latter paper provided a success framework for software development projects, it became necessary to set boundaries for the sample groups studied (Paper III). The second part then utilized the research results from the first one in order to study how Kanban (Paper IV, V, and VI) affects project success. The following sections summarize the research contribution of each paper while Figure 5.1 presents which research paper relates to which area.

5.1 Paper I – Discovering High-Impact Success Factors in Capstone Software Projects

Several isolated relationships can be deduced from the literature between a "project success" and, for example, "actions" of software development teams. These relationships come from several sources with a wide difference in the context where they were identified.

The findings of Paper I show that certain disciplines and theories can be used to determine project success. However, the field of software development still lacks a holistic model for project success in software development (Ikonen and Abrahamsson, 2010). Overall, the findings of the pilot study imply that by knowing what to look for, at least some success signs are evident from the beginning of the projects. Due to the fragmented field of software development success, this finding led the author to operationalize the concept of success in software engineering projects (Chapter 5.2).

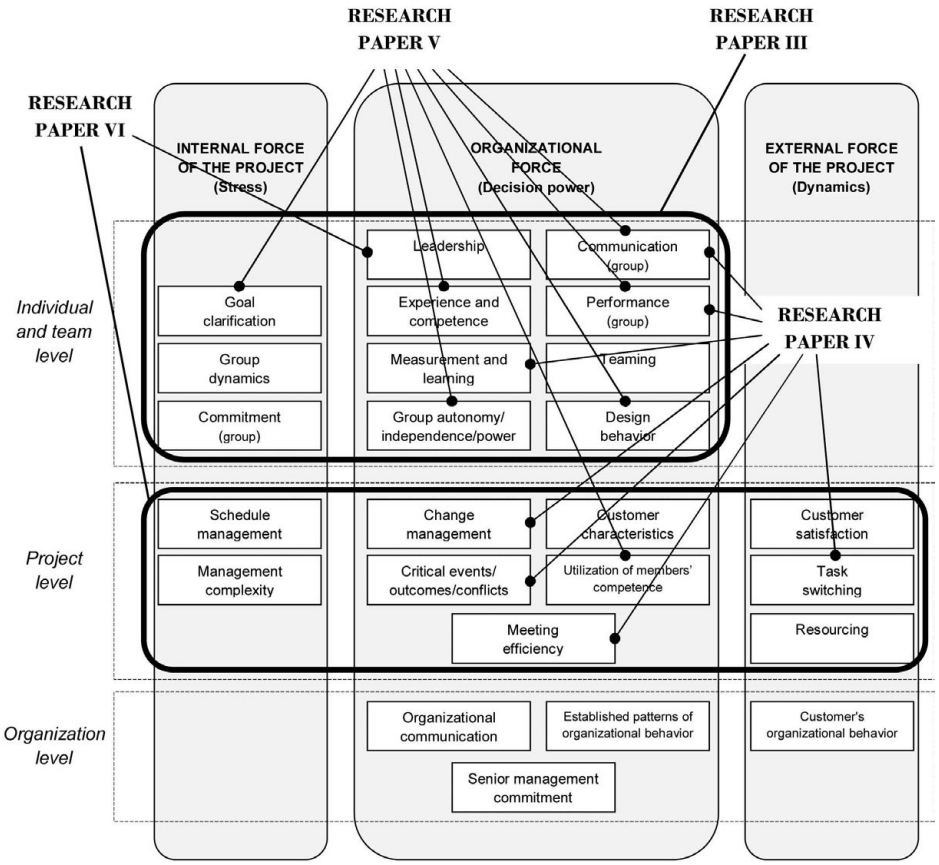

Figure 5.1: The relationships of the research papers to our framework presented in Paper II.

5.2 Paper II – Operationalizing the Concept of Success in Software Engineering Projects

Improving software development projects requires understanding the components of project success holistically. The literature of software engineering projects has focused on identifying isolated success factors even though the issues behind the success lie on multidimensional constructions with multidisciplinary factors. Moreover, research on project success regarding software engineering is still fragmented. While the term of project success is understood in a variety of ways (Agarwal and Rathod, 2006), some definitions taking the customer into account can be agreed to have an established position. Examples of such definitions are the four dimensions of project success of Shenhar et al. (2001) and a definition for project success by PMI (2008).

The organizations of software development have not had a standard framework to follow systematically. While the concept of success is often overlooked, the concept of project management is equally little addressed theoretically. Due to the reasons addressed here and in Chapter 3.1, Paper II conceptualizes the project success of software engineering by suggesting a success framework which organizes the fragmented area of the literature. The framework proposed and empirically evaluated in the paper can explain project success and help practitioners to find targets for improvement in projects. Thereby, the framework indicates how to provide a valuable tool for the organizations to improve their own projects and to consult their customers, as well.

Paper II shows several potential reasons for project failures. The trails of these reasons led to an inappropriate process model that represents a conventional, plan-driven approach to software development. The reasons showed include the lack of shared leadership. This leadership is part of high-performing teams (Blanchard et al., 2004, 12–13) and group dynamics. In other words, the findings encourage focusing on self-organization. Thereby, Paper III (Chapter 5.3) focuses on self-organizing teams in software development.

5.3 Paper III – Building Blocks for Self-organizing Software Development Teams

Self-organizing teams are considered to bring many advantages to organizations (Behnke et al., 1993; Cohen and Bailey, 1997; Janz, 1998). One of the key benefits is performance effectiveness (Cohen and Bailey, 1997)

but also customer satisfaction (Behnke et al., 1993; Jian'an, 2008; Winter, 1994). The main reason for the effectiveness is that self-organizing teams can react to problems quickly since the decision making is close to the problem (Moe et al., 2009a; Tata and Prasad, 2004). Instead of waiting for a manager's approval, the team has the authority to take necessary actions by itself (Moe et al., 2009a). This "empower the team" principle is part of Lean thinking as mentioned in Chapter 2.3.2.

Other advantages are positive changes in team member attitudes, including increased job satisfaction, stronger commitment to the organization, and trust toward management (Cohen and Bailey, 1997). These are suggested in many studies (see e.g., Guzzo and Dickson (1996); Janz (1998); Moe et al. (2009a)). One reason for the attitude changes is that self-organizing teams stimulate participation and commitment, which make the employees care more for their work (Fenton-O'Creevy, 1998; Moe et al., 2008). Moreover, self-organization causes positive behavioral outcomes including the level of absenteeism, turnover, and safety (Cohen and Bailey, 1997), which is reported in many studies, as well (Guzzo and Dickson, 1996).

Some research results are, however, contradictory (Guzzo and Dickson, 1996; Tata and Prasad, 2004). For example, some results indicate no connection between empowerment and success or that project performance is not increased (Reilly and Lynn, 2003). This contradiction means that self-organization is not a panacea, and just calling a group self-organizing does not automatically translate into better performance. The organizational context like the reward system, leadership, training, available resources, and the structure of the organization influence how teams can self-organize and perform (Cohen and Bailey, 1997; Tata and Prasad, 2004). The contradictory research results indicate a need to better understand what makes a self-organizing team successful (Fenton-O'Creevy, 1998).

Paper III contributes to the growing body of empirical understanding in the area of how to build a self-organizing software development team. A novel model for building self-organizing teams was constructed and empirically evaluated. Based on the empirical evidence, the research model in Paper III provides a framework to understand how self-organizing software development teams can be built. The results show that autonomy together with communication and collaboration are the major components for building a self-organizing software development team. The other building blocks are shared leadership, learning, redundancy, and team-orientation.

The findings provide an increased understanding of the benefits of self-organizing teams in software development. This understanding enables

studying the context wherein these kinds of teams are working. The process needs to function under a process model, which creates a need for further knowledge of how a process model impacts on the work. Kanban has been chosen based on the reasons presented in Chapter 2 and on the practical example given in Chapter 3. Moreover, due to the problems of the conventional, plan-driven software development found in Paper I and II, the rest of the research papers (Paper IV, V, and VI) examine impacts of Kanban on software development by conducting empirical studies in projects that consist of self-organizing teams.

5.4 Paper IV – Exploring the Sources of Waste in Kanban Software Development Projects

In order to function efficiently, self-organizing teams need a method to support a project's operational efficiency. Kanban (Chapter 2.5) is considered to be such a method that supports Lean thinking (Chapter 2). As stated in Chapter 2.2.2, eliminating all kinds of waste is a key trait of Lean thinking. One of the Lean production tools is the Kanban way of managing production operations that has been applied to software production as a project management process model (Hiranabe, 2008). Following this line of Lean thinking led us to conduct a study to find out whether Kanban is able to reveal waste in software development. This question is based on a thinking tool of "seeing waste" suggested by Poppendieck and Poppendieck (2003). Therefore, Paper IV explores different sources of waste in a case study of a Kanban-driven software development project.

As a result, all potential sources of waste proposed in the research model of the paper were found in practice at varying levels. Finding waste, however, did not significantly contribute to explaining project success. The good success of the study object is a probable reason explaining why only a small amount of waste was found. Whether this finding is the direct result of the application of Kanban, experience of the team leader and developers or customer pressure, remains harder to pinpoint. Yet, it is argued that waste represents elements that restrain the progress of projects, which endangers their chances for success. Paper IV demonstrates that zero waste is not a requirement for a successful project.

Thus, when using the seven possible elements of waste (Chapter 2.2.2) as the lenses for an analysis, it is likely that most of them are apparent in any software development project. What makes a difference is the attempt to minimize their impact or existence. This is where Kanban is useful. Even if Kanban strives for minimizing the non-value-adding work, waste

may still creep in as Paper IV shows. Therefore, by applying Kanban as a means to organize the development, attention must be paid to eliminating sources of waste.

Overall, eliminating waste is only a part of a project's operational efficiency supported by Kanban. While Paper IV tracks sources of waste in a Kanban-driven context, Paper V (Chapter 5.5) focuses on the impact of Kanban on software project work.

5.5 Paper V – On the Impact of Kanban on Software Project Work

Benefits of Kanban-way scheduling are reduced inventory (simultaneous WIP), improved flow, prevented overproduction, operations-level control, visualized schedule and management of the process, improved responsiveness to changes in demand, minimized risks of inventory obsolescence, and increased ability to manage the supply change (Gross and McInnis, 2003, 4). As a result, the Kanban method (Chapter 2.5) attempts to lower production costs, increase quality, and accelerate lead time (Liker, 2004). Meanwhile, inventories and problems caused by sudden changes will become more apparent. Nevertheless, while Kanban attempts to clarify the workers' awareness of current production issues and forthcoming tasks, it does not recommend any particular project phases, milestones, or partitioning tasks. Due to this liberty, it is up to each project team to build and customize the appropriate practices for its project.

Paper V investigates how Kanban influences software development and, consequently, project work. More specifically, this management-related research question in the paper was set as follows: What are the perceived impacts of Kanban on software project work? This is particularly important as projects "are conceived and completed by people" (Howell et al., 2004). Howell et al. (2004) argue that projects still lack the theoretical foundation that connects leadership and people aspects holistically. Kanban, as a method, empowers people with a minimum set of required rules to follow. In order to analyze the effects of Kanban upon software project management, Paper V comprises nine literature-based aspects of project work and the expected influences of Kanban on them. These nine aspects are documentation, problem solving, visualization, understanding the whole, communication, embracing the method, feedback, approval process, and selecting work assignments. Kanban was found to support all but *understanding the whole* and *feedback*.

Kanban affects the work aspect of **the amount of documentation**. In

Kanban, the customer "pulls" results from the software developers rather than the developers "pushing" results to the customer. Therefore, documents are not produced in Kanban without the need expressed by the customer.

Regarding **problem solving**, the Lean approach requires that problems are solved immediately and completely in order to prevent the same kinds of problems in the future (Liker, 2004). Using the Kanban method can reveal problems almost immediately after their occurrence (Ladas, 2009). **Visualizing** the development process helps the developers to see the state of that process: how much work has already been done and how much is yet to be done. The Kanban method is visual because all the work is visualized on the Kanban board. By looking at this board, everyone can see how a single task as well as the whole project is progressing. On the other hand, the Lean principle of "seeing the whole" partially exists in Kanban, since the Kanban board shows the work in progress, what needs to be done, and what has been done already. However, the board does not tell us how much work there is altogether. **Communication** in Kanban is important and it should be free and open, which follows the Lean approach (Kajaste and Liukko, 1994).

Related to **embracing the method**, the Kanban method is intuitive to understand and gives rather free hands to the developers to do their work: the only rule regarding this is that the workflow must be visualized. Lean principles emphasize that the best expertise is at each workstation and no massive **approval processes** need to be performed. There is neither need nor time to carry commands and instructions back and forth in the organization. Finally, the Lean approach prioritizes the satisfaction of the customer through a valuable product. Thereby, **selecting working assignments** can be made by developers themselves according to the Lean approach. There, the work is self-organizing.

Based on the empirical evaluation, the most significant influences stem from the inherent visualization of Kanban. Besides, in order to function well, Kanban requires visualizing the progress. This visualization helps in controlling the project activities in flexible yet coherent ways by relying on the intuition of the team members and emergence. Visualization was even found to motivate the team members. Another supporting trait is that the non-prescribed structure of the Kanban board was found to encourage the team members proactively to think about what the workflow should be like. The simplicity of the Kanban model allows situational adaptation, which is crucial nowadays in the dynamic environment of continuously and rapidly changing software development.

In contrast, the evidence implies that Kanban is not all-encompassing, and it is not sufficient for managing all the dimensions of software projects. For instance, while the Kanban board helps in detecting potential problems and bottlenecks early, it requires additional practices to actually solve them. Moreover, "seeing the whole" may still be difficult in particular with large, complex system projects with a simple Kanban board alone. That is, the Kanban needs supportive practices and contextual linking, for example, to incorporate customer feedback.

The empirical evidence hinted that the adoption of Kanban in use is quite straightforward even though continuous modifications and adjustments to the actual use of the board are done. While Kanban offers means for coordination purposes, the inherent simplicity of the approach supports the perception of creative freedom necessary in software development. The need for such freedom seems often to be undermined or overlooked by project management tool providers.

As shown in Paper III, shared leadership is a fundamental part of self-organizing teams. A fundamental part of Lean thinking, in its turn, is eliminating waste (Chapter 2.2.2). While Paper V focuses on Kanban from the viewpoint of management and shows benefits for software project management, a fundamental question remains: among all self-organization and Lean thinking, is leadership and management still necessary or have they become waste? Paper VI (Chapter 5.6) examines this question.

5.6 Paper VI – Leadership in Kanban Software Development Projects

Useless actions and work in software development projects do not increase the value for the customer, as the literature addressed in Chapter 2. While getting rid of such waste may sound simple, even recognizing the waste is considered a challenging issue. Once recognized with its causes, projects are more aware of the signs of waste: the pitfalls are avoidable by knowing their reasons. On the other hand, self-organization and empowering the teams emerge in a modern Kanban-driven software development project (Paper III, IV, and V). This makes it relevant to ask whether sacrificing project resources for leadership[1] adds any value even though Kanban, according

[1] In Paper VI, the term *leadership* refers to the definition of a leader's directive behavior by Blanchard (2001). He defines leadership in general as the pattern of behavior leaders use to influence others and perceived by those being influenced. Directive behavior of a leader in the terms of Kotter (1996, 25–26) is called *management*. We use Blanchard's term "leadership" in Paper VI to emphasize that management (in Kotter's terms) is

to previous studies, benefits management. Hence, Paper VI conducts a quasi-controlled experiment with two directive leadership settings in order to find out differences between the waste produced, its causes and effects.

The results supported the existence of one of the dual characteristics of Kanban: it allows working without a formal project manager in order to avoid waste but insufficient directive leadership creates waste. While some waste was not avoidable with either leadership style used in the experiment, causes and damages of the waste differed with different leadership. As a conclusion, the amount and significance of waste can be reduced with the right leadership in self-organized teams of Kanban software development projects.

5.7 Summary

This chapter summarizes the findings of the research conducted in the research papers. These empirical findings must be regarded somewhat tentatively due to the limited sample. The ensemble they formulate, however, provides knowledge for theory as well as for practice (Chapter 6).

By knowing what to look for, some success signs are evident from the beginning of projects (Paper I). This awareness creates a motivation being able to recognize signals from problems in the early stages of projects in order to prevent failures. In the empirical evaluation of Paper II, the project success framework (Figure 3.1) revealed a variety of critical issues that should be focused on from the outset of projects. Due to drawbacks of plan-driven development, Paper III searched for a solution for more efficient work from self-organizing. This self-organization combined with Kanban (Paper IV, V, and VI) was found to have a positive effect on software development projects.

insufficient for a skillful leader.

Chapter 6

Discussion

The discussion will be built on the two key implication areas of the research at hand. These are the theoretical implications for research and practical implications for practitioners including developers and managers of software development projects.The research presented in this study supports increased attention to be given to the Lean thinking that is adapted to software development processes. Aggregating the different perspectives and examining the results through the research papers leads to stronger results in terms of research validity than looking at each paper from an isolated viewpoint. Chapter 6.1 discusses the findings of the thesis and implications of the papers for research while Chapter 6.2 takes a look at the practical side.

6.1 Implications for Research

From the empirical viewpoint, this thesis has implications for research. These implications are addressed in the following.

Paper I and II empirically found certain success factors regarding plan-driven software development projects. They demonstrated drawbacks of the conventional development approach which already have been reported in the literature. This was expected due to the literature-based research model built in the study. Supportive evidence has been presented, for example, by Hutchings et al. (1993) and Bullen and Rockart (1986) regarding group dynamics, and by Curtis et al. (1988), Jensen and Scacchi (2005), and Teasley et al. (2000) regarding critical success factors of the software development projects.

The thesis contributes to the area of project success in software development. Chapter 3.2 combined with Paper II demonstrated how plan-driven

projects can be analyzed with the aid of Lean thinking. The research of the thesis showed that the general framework of software development project success (Figure 3.1) can be used as a basis for such analysis. This can be seen to be important for the plan-driven approach. Based on the increased awareness of what to look for, the Lean software development principles (Chapter 2.3.2) and waste categories (Table 2.1) help us to find the targets of improvements. For instance, the plan-driven approach does not explicitly encourage eliminating task switching in contrast to Lean software development (Poppendieck and Poppendieck, 2003). This was pointed out by the literature addressed in Chapter 2.2.

On the other hand, one project-related problem based on the interview data of Paper I and II that we found in the plan-driven projects but not in the Kanban-driven projects is the meaning of unawareness at the beginning of a project: In the plan-driven projects, this unawareness mostly led to accumulated problems in the phase after the planning (requirement analysis) phase, i.e., in the designing phase. In that phase, it typically turned out that requirements were too tricky or inappropriate when it came to the schedule that had already been approved and committed. This is aligned with the literature (Abrahamsson et al., 2010; Poppendieck and Poppendieck, 2007; Takeuchi and Nonaka, 1986). Our contribution to the matter stems from the finding that the Lean thinking, including Kanban, can be a valuable tool when software development projects are being analyzed with the aid of the general success framework (Paper II) presented in Figure 3.1.

Paper II operationalized the concept of success in software engineering projects and revealed problems that occur in the projects. Comparing these problems with the problems that occurred in Kanban-driven projects (Paper IV, V, and VI) creates a theoretical need to establish whether problems occurring in Kanban-driven projects are similar to the plan-driven ones. Particularly, it can be studied whether the set of problems in Kanban-driven projects is only a subset of the set in plan-driven projects. If so, this could mean that Kanban prevents problems in projects of the conventional approach. On the other hand, it should be evaluated, whether the Kanban-special problems could be avoided with the plan-driven approach. One difference between the conventional and Lean approach is the duration of the feedback loop, which is aligned with the literature (see, e.g., Poppendieck and Poppendieck (2007); Reinertsen (2009); Sommerville (2007)). Finding mistakes and misunderstandings rapidly prevents them from accumulating more serious issues, as the literature has addressed regarding the agile approach (Chapter 2.4). The empirical evidence of our research

showed the importance of the short duration of the feedback loop. The contribution was that Kanban was found to enable short feedback loops but it was up to the teams whether to utilize this possibility.

Yet another contribution for research that originated from our observation is that visualization motivates team members (Paper V). This observation – that a process model can serve as a motivator for the work – appears not to have been paid much attention in the literature. Empirical evaluation is needed to determine how motivating different process models can be and why.

Overall, the thesis has increased our understanding about adaptability of Lean-based Kanban in the context of software development. While the research focused particularly on Kanban-driven self-organizing teams, it is important to understand that self-organization alone is not the solution. Certain Lean or agile practices are needed when the idea of self-organization is applied to Kanban projects (Paper III, IV, V, and VI).

6.2 Implications for Practice

From the empirical viewpoint, this thesis has several implications for practice on two approaches of the software development process. First, it contributes in the area of Lean software development by supporting the argument that Kanban can be applied to the software development. Second, it supports analyzing plan-driven projects and, thereby, provides hints for practitioners on what to look for to avoid certain problems in the projects. These implications are addressed in the following.

6.2.1 Implications for the Kanban-driven Approach

Kanban's adaptability for manufacturing and production has been studied in the literature as addressed in Chapter 2. Kanban executes the Lean thinking in practice (Becker and Szczerbicka, 1998; Chai, 2008). Also, it is one of the key operation management tools in Lean manufacturing (Liker, 2004, 176). There is a strong practitioner-driven movement supporting the idea of the use of Kanban in software engineering (Hiranabe, 2008; Shinkle, 2009; Shalloway et al., 2009). The outcomes of this applying of Kanban are expected to be high in software development, as they have been in manufacturing.

The research in this thesis contributes to that expectation by supporting the argument that Kanban is applicable for software development (Paper IV, V, and VI). Demonstrated by Paper IV and VI, Kanban's aim at visualizing the progress, limiting the amount of simultaneous work-in-progress,

and measuring the lead-time, there is evidence that Kanban prevents waste. Based on the empirical evidence, Kanban benefits software development due to its visuality, intuitiveness, and simplicity that not only reveal waste (Paper IV and VI) but remove impractical operations, such as unnecessary documentation (Paper V) of the conventional software development, and help a project to adapt to continuously changing situations (Paper IV, V, and VI). Similarities to the limited literature of the area can be found. Middleton and Joyce (2011), for instance, show that the Kanban-driven Lean approach improves the software development process. Furthermore, the research in this thesis brought out the importance of limiting WIP since too large WIP limit numbers caused task switching and partially done work, in other words, waste (Paper VI).

After all, when people do not know what they are looking for, finding the answer is unlikely. Particularly Paper IV found waste in each of the seven waste categories (Chapter 2.2.2), suggested by Poppendieck and Poppendieck (2003). Hence, practitioners by being aware of these seven kinds of waste get an idea of what they should look for in their projects in order to eliminate waste. By revealing waste, significant opportunities in terms of saving resources and accelerating cycle time may be reached for practical use. Therefore, it can be concluded that Kanban executes the most important principle of Lean software development, eliminating waste. This ranking of the principle has been suggested by Poppendieck and Poppendieck (2003) and addressed in Chapter 2.3.2. The fundamental idea behind this ranking is to highlight the producing value (Chapter 2). Based on this highlighting, Kanban supports the production of value. Outside software development, evidence of this value-thinking already exists (Becker and Szczerbicka, 1998; Chai, 2008; Ohno, 1988). This thesis, in contrast, has focused on this argument from the aspect of software development.

Paper V demonstrated that the most significant influences stem from the inherent visualization of Kanban. This visualization helps in controlling the project activities in flexible yet coherent ways by relying on the intuition of the team members and emergence. Visualization was even found to motivate the team members as discussed regarding implications for research (Chapter 6.1). Practitioners can use Kanban to motivate team members. In addition to the visualization, the non-prescribed structure of the Kanban board can encourage team members proactively to think of what the workflow should be like. The simplicity of Kanban allows situational adaptation. Blanchard et al. (1996) consider this adaptation to be crucial in a present-day dynamic environment of continuously and rapidly changing

situations.

The comparison with the waterfall model made in Paper V suggested that Kanban requires experience to some extent to adjust and customize the setting. On the contrary, the threshold for deploying the waterfall model is considered to be low. When facing problems then, Kanban allows the team to make readjustments flexibly while the waterfall model may require a heavy re-planning or re-designs. For a customer, the waterfall model is also simple to get familiar with. Problems, however, are typically invisible for the customer until the end of the project. Kanban, instead, requires contribution from the customer in order to add value for him. Therefore, in addition to communication within projects or organizations, also good customer communication in software development projects is important (Beck, 1999). This was recognized in each paper. Moreover, we managed to track sources of waste and its causes and effects in Paper VI.

In contrast to the advantages of Kanban, the empirical evidence of Paper V and VI implies that Kanban is not all-encompassing, and it is not sufficient for managing all the dimensions of software projects. In other words, it requires additional practices to keep the projects performing appropriately. For instance, "seeing the whole" may be difficult in particular with large, complex system projects with a simple Kanban board alone. Middleton and Joyce (2011) report supportive *information radiators*[1], such as ideation pipeline and team performance indicators.

The inherent simplicity of Kanban supports the perception of creative freedom (Paper V), which can be risky: Paper VI showed that a Kanban-driven project without appropriate coordination and management increases the amount of waste. Thereby, practitioners should avoid optimizing the software development process too much in the sense of coordination.

For practitioners, Kanban's adaptability to software development means that its advantages compared to the conventional approach of software development can be utilized in a journey toward more successful outcomes of the projects. Meanwhile, it is important to understand that Kanban alone is not the answer to more successful projects. Furthermore, finding waste is not synonymous with successful projects (Paper IV). On the other hand, zero waste is not a requirement for a successful software development project (Paper IV and VI).

The implications discussed above are based on the study objects that

[1]In the team context, Cockburn (2002, 84) introduces the term information radiator as a large display of critical team information that is continuously updated and located in a spot where the team can see it constantly. Such a radiator shows readers information they care about without having to ask anyone a question. According to Cockburn (2002), this results in more communication with fewer interruptions.

have been conducted as self-organizing teams. Paper III suggested the building elements of autonomy, communication and collaboration, shared leadership, learning, team orientation, and redundancy. This has two practical implications. First, teams should have an understanding of practices required for the self-organization. Second, while process models regularly do not explain how self-organizing teams can be built, the combination of the six elements makes an attempt to do so. Since the study objects were self-organizing according to the definition, the applicability of the implications is scoped to concern self-organizing teams. Self-organization relates to Kanban and is part of Lean thinking as addressed in Chapter 1.1.

6.2.2 Implications for the Plan-driven Approach

The research in Paper I and II offers a model for practitioners to find targets for improvement in their plan-driven projects. In other words, the contribution of this part of the thesis is not just conducting theoretical models but suggesting that practitioners utilize them as self-test instruments. Thus, the idea is to create better awareness among the practitioners about the key elements that contribute to success or failure of the projects. Project success, in terms of project efficiency and impact on the customer, can be seen after the project ends. However, economical tension requires a business decision to abort projects with little or no chance to succeed as early as possible. Without search or anticipation (Paper I), these decisions are difficult to make until the end of a project. This is where the project success framework (Paper II) benefits practitioners.

Monitoring ongoing projects enables detecting potential issues that will restrain the projects from going forward. The set of questions, introduced in Paper I, guide practitioners to increase their awareness of the project situation during the progress. This way, by tracking reasons for the project issues, we can reveal problems that can be eliminated before they accumulate to more serious problems. Ikonen and Abrahamsson (2010) support this claim. The critical success factors found in Paper I are quite similar to the literature (Addison and Vallabh, 2002; Curtis et al., 1988; Keil et al., 1998). Here, the contribution of the thesis is the way they are used to detect signals of threads in the projects beforehand.

Together, Paper I and II demonstrated that the area of project success in software development is fragmented. Paper II suggested a research model that organizes the area. As a result, the framework (Figure 3.1) illustrates the complex nature of the area. The three forces of the project success framework (i.e., internal, organizational, and external) affected projects help us to think of the phenomena of projects from the viewpoint

of each force. By connecting this viewpoint to each level (i.e., individual, team, project, and organization) the framework helps to increase practitioners' awareness about the holistic picture of software development projects. From that point of view, reasons for project failures may be better understood. By determining the framework as a profile-type (Chapter 3.2) helps practitioners to understand further reasons why software development projects cannot fully be under control.

Chapter 7

Conclusions

This thesis, after having proposed a framework for project success in software development, explored impacts of Kanban to software development projects. This exploration increased our understanding about adaptability of Lean-based Kanban in the context of software development. According to the research in this thesis, it can be stated that Kanban has potential for software development and can be applied in this context. The research papers of the thesis, refereed, international journal and conference publications, provided information for both research and practitioners as discussed in Chapter 6.

Since Lean principles encourage project teams toward self-organization, the thesis focused on self-organized teams in Kanban-driven projects. Thereby, this thesis also examined the building of self-organized teams for the purpose of exploring Kanban. In contrast to conventional software development, Lean practices were shown to take into account many such issues that are ignored by the conventional approach. These advantages were shown to benefit projects by increasing the fluency of the projects. In contrast to agile software development, Lean practices were found to concretely emphasize eliminating waste and seeing the whole. Moreover, Lean software development was found to focus on adding value to customers in a practical way. The literature has shown many advantages of the agile approach when compared with the conventional approach. The comparison of the Lean approach with the agile approach including the literature addressed in this thesis, however, suggested that the Lean approach when applied to software development could be even more useful than the agile one.

Regardless, applying the Lean thinking to software development is not a guarantee of similar success as in the field of Lean manufacturing. This is because laws and principles of Lean manufacturing differ from the Lean

software development (Chapter 2): problems in pipeline processing related to physical material flows and machine or worker activities are easier to see than in software development wherein waste is rather abstract than visible.

Chapter 7.1 answers the research questions presented in Chapter 1.1 while Chapter 7.3 opens up new avenues for future work by suggesting some further research.

7.1 Answering the Research Questions

Chapter 1.1 presented the research question and divided it into three research sub questions. In the following, we answer them with a summary of the results.

RQ1 How does Kanban impact on software development projects?

Answering this research question requires understanding of

(1) what prevents or promotes projects going forward

(2) the concept of self-organization due to the literature aspect that recommends self-organization for the Lean project context to be explored for comparability of the results between the case projects

(3) influences of Kanban on the progress of projects from the viewpoints of waste, project work, and management.

Based on this understanding, we claim that Kanban has a positive impact on software development when used appropriately. Kanban was found to help in revealing waste, which can save the resources and time of software development projects. Visualizing the progress was found to aid in controlling project activities in flexible yet coherent ways and even to motivate software development teams. In addition, Kanban was also found to enable the project teams rapidly to adapt to continuously changing situations. More detailed answers for the three research sub questions, RQ1.1, RQ1.2, and RQ1.3 are given as follows.

RQ1.1 Which constructs of software development projects affect the outputs?

Since Paper I showed the existence of evidence of some success signs from the beginning of software development projects, a need for more specific

study arosed. Paper II operationalized the concept of success in software engineering projects. As a result, it suggested a general framework for software project success, which was then linked to the context of Lean software development. The purpose of the framework is to help in understanding, in addition to project-related problems, also reasons and effects of these problems. The framework organizes the fragmented area of project success in software engineering

The empirical evaluation of the framework (Paper II) revealed the failure potential of the software development projects. The implicit use of the framework in Paper IV, V, and VI strengthened its validity since factors for partial failure of the case projects were taken into account in the framework. Based on this evidence, the research results suggest that determining success of a software development project in the midst of failure can be done with the aid of the framework.

RQ1.2 What are the key elements of self-organizing teams?

Paper III recognized six general elements of self-organization in Lean or agile practices. The two foundational elements were autonomy (i.e., empower the team and have a champion for it) and communication and collaboration (i.e., intimate and open customer relationship, working together in open workspace, daily information sharing, and progress visualization). Self-organization without autonomy is only symbolic meaning that if a team has no autonomy, it really cannot act like a self-organizing work unit. Communication and collaboration, in its turn, is important for the other elements of self-organizing teams. The additional four elements were shared leadership (i.e., managing by lead-and-collaborate and having cross-functional teams), learning (having short iterations, continuous feedback, and progress measurement, for example), team orientation (i.e., clear prioritizing and having the team participate in work planning and goal setting), and redundancy (i.e., sharing responsibility of work and agreeing on uniformity).

The literature reports that transformation of a work group into a self-organizing work team takes years. Our empirical data, however, indicated that building a self-organizing software development team is possible, at least to some extent, in as little as seven weeks. In this sense, self-organized teams can be built by following the six general elements mentioned above.

RQ1.3 What are the salient characteristics of Kanban that affect the progress of software development projects?

Kanban, when applied to software development as a process management model, is supposed to have positive impacts when compared with agile or conventional process models. Paper IV, V, and VI explored influences of Kanban on software development projects.

Regarding the viewpoint of waste in Paper IV and VI, the Kanban method helped to prevent waste, such as partially done work and task switching. This helps in revealing problems quickly, which, on its part, allows project teams to focus on the actual progress instead of fixing consequences caused by unrevealed problems. The Kanban-supported pull method forced both customer and team members to think about what would be the most important tasks that should be carried out next. This prioritizing reduced the amount of unnecessary features of the products being developed.

The research found waste in each of the waste categories suggested by Poppendieck and Poppendieck (2003), which indicates two things. First, discovering waste in Kanban-driven software development projects is possible. This is important since waste in software development, in contrast to manufacturing, is considered abstract and invisible. Second, being aware of the concrete waste categories of Lean thinking adapted into software development helps in understanding what we are looking for. Then the target of search and improvements becomes possible. On the other hand, some waste was not related to Kanban. For example, the waiting the research found was also caused by external reasons, such as waiting for the customer.

The empirical results of Paper V supported the fact that Kanban, in order to function well, requires visualizing the progress. This visualization helped in controlling the project activities in flexible yet coherent ways. Visualization was even found to motivate the team members. The non-prescribed structure of the Kanban board, in its turn, encouraged the team members proactively to think of how the workflow is going.

The simplicity of Kanban allows situational adaptations, which is crucial in today's dynamic environment of continuously and rapidly changing software development. Regardless, Kanban was found to be insufficient for managing some dimensions of software projects. While this fact does not dim Kanban's power as an applied process management model for software engineering, this shortcoming is important to be aware of in practice.

Finally, the empirical data of Paper VI demonstrated that management (i.e., the leader's directive behavior) is fundamental in order to make Kanban-driven software development projects successful. In other words,

investigating resources for good management is not waste even in the Lean context.

7.2 Limitations of the Thesis

The validity of the results presented in this thesis is limited by the fact that the evaluation of the project success framework has been done with the plan-driven setting. This, however, is also an advantage. The conventional software development is considered mature, which enabled us to examine whether the framework can reveal problems in the projects. Besides, a corresponding setting in the context of Lean software development would have been inappropriate due to its immature status. Thereby, examining the plan-driven setting provided a good starting point for the research in the thesis. The themes and constructs of the framework were used to analyze Lean-based Kanban projects, which showed the expressive power of the framework in the Lean context, as well. In addition, the questionnaires designed for the framework mostly concerned common matters for both software development approaches. In this sense, the framework functioned in both conventional and Lean context as shown: the relevant areas of success are fundamental also regarding Lean software development.

The empirical data contained student but also industrial study objects. It is quite well established that when one seeks to set a trend, the use of students is quite acceptable (Tichy, 2000). Tichy (2000) uses a method comparison as a specific target of study where the use of students is a valid approach. Others have made similar suggestions. Höst et al. (2000), for example, conclude that students are indeed relevant when considering experimentation in software engineering. Kitchenham et al. (2002) remind us not to look down on studies focusing on gathering empirical data from student experimentation or projects. Madeyski (2010) also gives strong argumentation for the benefit of using students as study objects. His specific study focused on test-driven development and pair programming. We do not maintain that our findings regarding the studies conducted in Software Factory are one-to-one with the industry but rather that given the specific circumstances, we indicate that there is a trend revealing impacts of Kanban on the projects when searching for a particular set of indicators. In addition, Arisholm and Sjøberg (2004) argue that the programming skills of Master level students and senior programmers in the industry can be considered equal. The small number of case studies can be considered to be another limitation. Tichy (2000), however, reminds us that the novel approach to the research question justifies this number.

7.3 Future Research

This thesis has intentionally focused on the individual and team level since there lies much potential to improve the software development process. Further research is needed to examine project and organizational levels and how these levels interact with the individual and team levels. More concrete understanding of the flow related to Lean software development is needed, as well. Yet, due to the small amount of empirical evidence in the research area, it cannot be determined whether an iterative or non-iterative approach is more effective and in what sense.

According to the preliminary research results (Chapter 5.1), project success can be anticipated by revealing mistakes, conflicts, and other restraining elements in projects. Hypothetically, the less restraining elements there are, the more successful the project will be. Nevertheless, this requires further experimentation in real-life settings. Besides, the thesis confirms that zero waste is not a requirement for a successful project. Moreover, inspired by the general framework of the project success, it could be studied whether Lean thinking has more expressive power in analyzing projects and their problems than does the plan-driven approach.

Due to insufficiencies of Kanban reported in this thesis, further research should examine what supportive practices extend holistically without limitations, such as hierarchy-related, in organizations. While it is required more research in the path toward more successful, Kanban-driven Lean software development, this doctoral thesis contributes by offering building blocks for that path.

References

Abrahamsson, P. (2000). Is management commitment a necessity after all in software process improvement? In *Proceedings of the 26th EUROMICRO Conference*, pages 2246–2253, Los Alamitos, CA, USA. IEEE.

Abrahamsson, P. (2010). Unique infrastructure investment: Introducing the Software Factory concept. *Software Factory Magazine*, 1(1):2–3.

Abrahamsson, P., Oza, N., and Siponen, M. (2010). Agile software development methods: A comparative review. In Dingsøyr, T., Dybå, T., and Moe, N. B., editors, *Agile Software Development*, pages 31–59. Springer Berlin Heidelberg.

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis*. Number 478 in VTT Publications. VTT Technical Research Centre of Finland, Espoo, Finland.

Addison, T. and Vallabh, S. (2002). Controlling software project risks: An empirical study of methods used by experienced project managers. In *SAICSIT '02: Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 128–140. South African Institute for Computer Scientists and Information Technologists.

Agarwal, N. and Rathod, U. (2006). Defining 'success' for software projects: An exploratory revelation. *International Journal of Project Management*, 24(4):358–370.

Alwardt, A., Mikeska, N., Pandorf, R., and Tarpley, P. (2009). A lean approach to designing for software testability. In *Proceedings of the IEEE AUTOTESTCON '09 Systems Readiness Technology Conference*, pages 178–183. IEEE.

Anders, D. (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. Prentice Hall, Upper Saddle River, New Jersey, USA.

Anderson, D. J. (2010). Business drivers for Kanban adoption. In *Lean Software & Systems Conference 2010*, pages 7–14, Atlanta, Georgia, USA. Lean Software & Systems Consortium.

Arisholm, E. and Sjøberg, D. (2004). Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):521–534.

Basili, V. and Turner, A. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, 1(4):390–396.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, Massachusetts, USA.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Flower, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. http://agilemanifesto.org/ [4-Feb-2011].

Becker, M. and Szczerbicka, H. (1998). Modeling and optimization of Kanban controlled manufacturing systems with GSPN including QN. In *International Conference on Systems, Man, and Cybernetics '98*, volume 1, pages 570–575 vol.1. IEEE.

Behnke, L., Hamlin, R., and Smoak, B. (1993). The evolution of employee empowerment. *IEEE Transactions on Semiconductor Manufacturing*, 6(2):143–155.

Bertelsen, S. and Koskela, L. (2003). Avoiding and managing chaos in projects. In *Proceedings of the 11th Annual Conference of the International Group for Lean Construction (IGLC-11)*.

Bertelsen, S., Koskela, L., Henrich, G., and Rooke, J. (2006). Critical flow: Towards a construction flow theory. In *Proceedings of the 14th Annual Conference of the International Group for Lean Construction (IGLC-14)*.

Blanchard (2001). *Situational Leadership® II – The Article*. The Ken Blanchard Companies.

Blanchard, K., Carew, D., and Parisi-Carew, E. (1996). How to get your group to perform like a team. *Training and Development*, 50(September):34–37.

Blanchard, K., Carew, D., and Parisi-Carew, E. (2004). *The One Minute Manager Builds High Performing Teams*. HarperCollinsPublishers, London, UK.

Boddy, D. (2002). *Managing Projects: Building and Leading the Team*. Prentice Hall International UK Limited / Pearson Education.

Boddy, D. and Macbeth, D. (2000). Prescriptions for managing change: A survey of their effects in projects to implement collaborative working between organizations. *International Journal of Project Management*, 18(5):297–306.

Boehm, B. (1981). *Software engineering economics*. Prentice-Hall, New Jersey, USA.

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

Bonaccorsi, A. and Rossi, C. (2004). Contributing to OS projects: A comparison between individual and firms. In *Proceedings of the 4th Workshop on Open Source Software Engineering*, pages 18–22. IEEE Computer Society.

Brooks, F. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19.

Bullen, V. and Rockart, J. (1986). *A Primer on Critical Success Factors*. Dow Jones-Irwin, Homewood, Illinois, USA.

Chai, L. (2008). E-based inter-enterprise supply chain Kanban for demand and order fulfilment management. In *International Conference on Emerging Technologies and Factory Automation EFTA '08*, pages 33–35. IEEE.

Clegg, S., Waterson, P., and Axtell, C. (1996). Software development knowledge intensive work organizations. *Behaviour and Information Technology*, 15(4):237–249.

CMMI Product Team (2007). *CMMI for Acquisition, Version 1.2: Improving Processes for Acquiring Better Products and Services*. Number

CMU/SEI-2007-TR-017 in Series of Technical Reports. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

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

Cohen, S. and Bailey, D. (1997). What makes teams work: Group effectiveness research from the shop floor to the executive suite. *Journal of Management*, 23(3):239–290.

Conradi, R. and Fuggetta, A. (2002). Improving software process improvement. *IEEE Software*, 19(4):92–99.

Cumbo, D., Kline, D. E., and Bumgardner, M. S. (2006). Benchmarking performance measurement and lean manufacturing in the rough mill. *Forest Products Journal*, 56(6):25–30.

Curtis, B. (1994). A mature view of the CMM. *American Programmer*, 7(9):19–28.

Curtis, B., Hefley, B., and Miller, S. (2001). *People Capability Maturity Model ® (P-CMM®) – Version 2.0*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287.

Cusumano, M. (1991). *The Software Factory: An Entry for the Encyclopedia of Software Engineering*. Sloan School of Management, Massachusetts Institute of Technology.

De Meyer, A., Loch, C., and Pich, M. (2002). Managing project uncertainty: From variation to chaos. *MIT Sloan Management Review*, 43(2):60–67.

DeMarco, T. and Lister, T. (1987). *Peopleware*. Dorset, New York, USA.

Díaz, J., Garbajosa, J., and Calvo-Manzano, J. A. (2009). Mapping CMMI level 2 to Scrum practices: An experience report. In *Proceedings of 16th European Systems & Software Process Improvement and Innovation (EuroSPI 2009)*, pages 93–104. Springer.

Emiliani, B., Stec, D., Grasso, L., and Stodder, J. (2003). *Better thinking, better results: Using the power of lean as a total business solution*. Center for Lean Business Management, Kensington, Connecticut, USA.

Emiliani, M., Stec, D., and Grasso, L. (2005). Unintended responses to a traditional purchasing performance metric. *Supply Chain Management: An International Journal*, 10(3):150–156.

Ernest-Jones, T. (2007). *I.T. at the speed of business: A survey and whitepaper*. Hewlett-Packard Development Company and the Economist Intelligence Unit.

Espinoza, A. and Garbajosa, J. (2011). A study to support agile methods more effectively through traceability. *Innovations in Systems and Software Engineering*, 7(1):53–69.

Fagerholm, F. (2010). Psychometric measurements in software development. *Software Factory Magazine*, 1(1):12–13.

Fenton-O'Creevy, M. (1998). Employee involvement and the middle manager: Evidence from a survey of organizations. *Journal of Organizational Behavior*, 19(1):67–84.

Filho, W. P. P. (2001). Requirements for an educational software development process. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 65–68, New York, NY, USA. ACM.

Freeman, M. and Beale, P. (1992). Measuring project success. *Project Management Journal*, 23(1):8–17.

Fuggetta, A. (2000). Software process: A roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 25–34, New York, NY, USA. ACM.

Glass, R. L. (2006). The Standish report: Does it really describe a software crisis? *Communications of the ACM*, 49(8):15–16.

Goldratt, E. M. (1984). *The Goal: A process of ongoing improvement*. North River Press, Great Barrington, Massachusetts, USA.

Goodson, R. (2002). Read a plant – fast. *Harvard Business Review*, 80(5):105–113.

Greenfield, J., Short, K., Cook, S., and Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools*. Wiley.

Gross, J. M. and McInnis, K. R. (2003). *Kanban made simple: Demystifying and applying Toyota's legendary manufacturing process*. AMACOM, New York, NY, USA.

Gunson, J., de Blasis, J.-P., and Neary, M. (2003). *Leadership in real time: A model of five levels of attributes needed by a project manager in ERP implementations*. Number 03-15 in UG-HEC-CR. HEC Geneva, University of Geneva, Geneva, Switzerland.

Guzzo, R. and Dickson, M. (1996). Teams in organizations: Recent research on performance and effectiveness. *Annual Review of Psychology*, 47(1):307–338.

Hartmann, D. (2004). Interview: Jim Johnson of the Standish Group. http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS [5-May-2011].

Hazzan, O. (2003). Computer science students' conception of the relationship between reward (grade) and cooperation. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 178–182, New York, NY, USA. ACM.

Hilburn, T. B., Mengel, S., Bagert, D. J., and Oexmann, D. (1998). Software engineering across computing curricula. In *ITiCSE '98: Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education*, pages 117–121, New York, NY, USA. ACM.

Hiranabe, K. (2008). Kanban applied to software development: From agile to lean. http://www.infoq.com/articles/hiranabe-lean-agile-kanban [4-May-2011].

Höst, M., Regnell, B., and Wohlin, C. (2000). Using students as subjects: A comparative study of students and professionals in lead-time impact assessments. *Journal of Empirical Software Engineering*, 5(3):201–214.

Hovorka, D. S. and Larsen, K. R. (2006). Enabling agile adoption practices through network organizations. *European Journal of Information Systems*, 15(2):159–168.

Howell, G. A., Macomber, H., Koskela, L., and Draper, J. (2004). Leadership and project management: Time for a shift from fayol to flores. In *Proceedings of the 12th Annual Conference of the International Group for Lean Construction (IGLC-12)*, pages 22–29.

Humphrey, W. S. (1995). *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, USA.

Hutchings, T., Hyde, M. G., Marca, D., and Cohen, L. (1993). Process improvement that lasts: An integrated training and consulting method. *Communications of the ACM*, 36(10):105–113.

Ikonen, M. and Abrahamsson, P. (2010). Anticipating success of a business-critical software project: A comparative case study of waterfall and agile approaches. In Tyrväinen, P., Jansen, S., and Cusumano, M. A., editors, *Proceedings of the 1st International Conference on Software Business (ICSOB 2010)*, pages 187–192, Berlin Heidelberg. Springer-Verlag.

Jacobson, I. (2009). What they don't teach you about software at school: Be smart! In *Proceedings of the 10th International Conference on Agile Processes in Software Engineering and Extreme Programming XP 2009*, pages 1–4. Springer.

Janes, A. and Succi, G. (2009). To pull or not to pull. In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, OOPSLA '09, pages 889–894, New York, NY, USA. ACM.

Janz, B. D. (1998). The best and worst of teams: Self-directed work teams as an information systems development workforce strategy. In *SIGCPR '98: Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research*, pages 59–67, New York, NY, USA. ACM.

Järvinen, P. and Järvinen, A. (1996). *Tutkimustyön metodeista*. Opinpaja, Tampere, Finland.

Jensen, C. and Scacchi, W. (2005). Collaboration, leadership, control, and conflict negotiation and the netbeans.org open source software development community. In *HICSS '05: Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, Washington DC, USA. IEEE Computer Society.

Jian'an, C. (2008). Research on strategies and empowerment process to achieve self-management team. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pages 1–5. IEEE.

Jones, T. (1986). *Programming productivity*. McGrawHill, New York, USA.

Jurison, J. (1999). Software project management: The manager's view. *Communications of the AIS*, 2(September).

Kajaste, V. and Liukko, T. (1994). *Lean-toiminta*. Metalliteollisuuden Kustannus Oy, Tampere, Finland.

Karlsson, C. and Åhlström, P. (1996). The difficult path to lean product development. *Journal of Product Innovation Management*, 13(4):283–295.

Keil, M., Cule, P. E., Lyytinen, K., and Schmidt, R. C. (1998). A framework for identifying software project risks. *Communications of the ACM*, 41(11):76–83.

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., Emam, K., and Rosenberg, J. (2002). Preliminary guidelines for empirical research for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734.

Klein, H. and Myers, M. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quartely*, 23(1):67–94.

Kniberg, H. (2009). Kanban vs. Scrum: How to make the most of both. http://www.crisp.se/henrik.kniberg/Kanban-vs-Scrum.pdf [18-Jan-2010].

Koskela, L. (2000). *An exploration towards a production theory and its application to construction*. Number 408 in Series of Doctoral Theses. VTT Technical Research Centre of Finland, Espoo, Finland.

Koskela, L. and Howell, G. (2002). The theory of project management: Explanation to novel methods. In *Proceedings of the 12th Annual Conference of the International Group for Lean Construction (IGLC-10)*.

Kotter, J. P. (1996). *Leading change*. Harvard Business School Press, Boston, Massachusetts, USA.

Ladas, C. (2009). *Scrumban: Essays on Kanban Systems for Lean software development*. Modus Cooperandi Press, Seattle, WA, USA.

Larman, C. and Basili, V. (2003). Iterative and incremental developments: A brief history. *IEEE Computer*, 36(6):47–56.

Law, K. S., Wong, C.-S., and Mobley, W. H. (1998). Toward a taxonomy of multidimensional constructs. *The Academy of Management Review*, 23(4):741–755.

Licker, P. S. (1992). Setting quality sights on HR management in IS. In *SIGCPR '92: Proceedings of the 1992 ACM SIGCPR conference on Computer personnel research*, pages 109–116, New York, NY, USA. ACM.

Liker, J. (2004). *The Toyota Way*. McGraw-Hill, New York, NY, USA.

Liker, J. K. and Hoseus, M. (2008). *Toyota Culture: The heart and soul of the Toyota Way*. McGraw-Hill, New York, NY, USA.

Lui, K. and Chan, K. (2008). *Software Development Rhythms*. John Wiley & Sons, New Jersey, USA.

Madeyski, L. (2010). *Test-driven development: An empirical evaluation of agile practice*. Springer–Verlag, Berlin Heidelberg.

Mandić, V., Oivo, M., Rodríguez, P., Kuvaja, P., Kaikkonen, H., and Turhan, B. (2010). What is flowing in lean software development. In Abrahamsson, P. and Oza, N., editors, *Proceedings of the 1st International Conference on Lean Enterprise Software and Systems (LESS 2010)*, volume 65, Part 2 of *Lecture Notes in Business Information Processing*, pages 72–84, Berlin Heidelberg. Springer-Verlag.

Maylor, H. (2001). Beyond the GANTT chart: Project management moving on. *European Management Journal*, 19(1):92–100.

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

Middleton, P., Flaxel, A., and Cookson, A. (2005). Lean software management case study: Timberline inc. In Baumeister, H., Marchesi, M., and Holcombe, M., editors, *Extreme Programming and Agile Processes in Software Engineering*, volume 3556 of *Lecture Notes in Computer Science*, pages 1–9, Berlin Heidelberg. Springer.

Middleton, P. and Joyce, D. (2011). Lean software management: BBC Worldwide case study. *IEEE Transactions on Engineering Management*, PP(99):1–13.

Moe, N., Dingsøyr, T., and Dybå, T. (2008). Understanding self-organizing teams in agile software development. In *Proceedings or the 19th Australian Conference on Software Engineering (ASWEC 2008)*, pages 76–85. IEEE Computer Society.

Moe, N., Dingsøyr, T., and Dybå, T. (2009a). Overcoming barriers to self-management in software teams. *IEEE Software*, 26(6):20–26.

Moe, N. B., Dingsøyr, T., and Røyrvik, E. (2009b). Putting agile teamwork to the test: An preliminary instrument for empirically assessing and improving agile software development. In *Proceedings of the 10th International Conference on Agile Processes in Software Engineering and Extreme Programming XP 2009*, pages 114–123. Springer.

Morgan, J. M. and Liker, J. K. (2006). *The Toyota Product Development System: Integrating people, process, and technology.* Productivity Press, New York, NY, USA.

Morgan, T. (1998). *Lean manufacturing techniques applied to software development.* Series of Master Theses. Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

Mujtaba, S., Feldt, R., and Petersen, K. (2010). Waste and lead time reduction in a software product customization process with value stream maps. In *Proceedings of the 21st Australian Software Engineering Conference (ASWEC 2010)*, pages 139–148. IEEE Computer Society.

Nightingale, D. (2009). Principles of enterprise systems. In *Second International Symposium on Engineering Systems*, Cambridge, Massachusetts, USA. MIT.

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production.* Productivity Press, New York, NY, USA.

Oivo, M., Kuvaja, P., Pulli, P., and Similä, J. (2004). Software engineering research strategy: Combining experimental and explorative research (EER). In Bomarius, F. and Iida, H., editors, *Product Focused Software Process Improvement (PROFES)*, volume 3009 of *Lecture Notes in Computer Science*, pages 302–317. Springer Berlin / Heidelberg.

Oppenheim, A. (1992). *Questionnaire design, interviewing and attitude measurement.* Pinter Publishers, London, UK.

Oppenheim, B. W. (2004). Lean product development flow. *Systems Engineering*, 7(4):352–376.

Oppenheim, B. W., Murman, E. M., and Secor, D. A. (2010). Lean enablers for systems engineering. *Systems Engineering*, 14(1):29–55.

Parnas, D. and Curtis, B. (2009). Point/counterpoint. *IEEE Software*, 26(6):56–59.

Parnell-Klabo, E. (2006). Introducing lean principles with agile practices at a Fortune 500 company. In *Agile Conference (AGILE) 2006*. IEEE Computer Society.

Patton, M. Q. (1990). *Qualitative evaluation and research methods*. Sage Publications, Thousand Oaks, CA, USA.

Pedrycz, W. and Succi, G. (2007). *Fuzzy Logic Classifiers and Models in Quantitative Software Engineering*, pages 3142–3159. Idea Group, Inc., Hershey, Pennsylvania, USA.

Perera, G. and Fernando, M. (2007). Enhanced agile software development: Hybrid paradigm with LEAN practice. In *International Conference on Industrial and Information Systems (ICIIS) 2007*, pages 239–244. IEEE.

Petersen, K. (2010). *Implementing Lean and Agile software development in industry*. Number 2010:04 in Series of Doctoral Theses. Blekinge Institute of Technology, School of Computing, Sweden.

Pfadt, A. and Wheeler, D. J. (1995). Using statistical process control to make data-based clinical decisions. *Journal of Applied Behavior Analysis*, 28(3):349–370.

PMI (2008). *A Guide to the Project Management Body of Knowledge*. Project Management Institute, Inc., Pennsylvania, USA, 4th edition.

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

Poppendieck, M. and Poppendieck, T. (2007). *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley, Boston, Massachusetts, USA.

Pressman, R. (1997). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Cambrigde, UK, European adaptation edition.

Pugh, S. (1991). *Integrated Methods for Successful Product Engineering*. Addison-Wesley, Reading, Massachusetts, USA.

Ramasubbu, N. and Balan, R. K. (2009). The impact of process choice in high maturity environments: An empirical analysis. In *Proceedings of the 31st International Conference on Software Engineering*, pages 529–539, Washington, DC, USA. IEEE Computer Society.

Ravichandran, T. and Rai, A. (1994). The dimensions and correlates of systems development quality. In *SIGCPR '94: Proceedings of the 1994 computer personnel research conference on Reinventing IS: Managing information technology in changing organizations*, pages 272–282, New York, NY, USA. ACM.

Reilly, R. and Lynn, G. (2003). Power and empowerment: The role of top management support and team empowerment in new product development. In *PICMET '03: Portland International Conference on Management of Engineering and Technology Technology Management for Reshaping the World 2003*, pages 282–289, Portland, OR, USA. Portland State University.

Reinertsen, D. G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, Redondo Beach, CA, USA.

Robson, M. (1993). *Problem Solving in Groups*. Gower, Aldershot, Hampshire, UK.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. In *Proceedings of IEEE WESCON*, pages 1–9. IEEE Press.

Scacchi, W. (1984). Managing software engineering projects: A social analysis. *IEEE Transactions on Software Engineering*, SE-10(1):49–59.

Schein, E. (1988). *Process Consultation*. Addison-Wesley, Reading, Massachusetts, USA.

Schwaber, K. (1995). Scrum development process. In *Business object design and implementation workshop*, OOPSLA '95, New York, NY, USA. ACM.

Schwaber, K. and Beedle, M. (2002). *Agile Software Development with Scrum*. Prentice-Hall, Upper Saddle River, New Jersey, USA.

Seaman, C. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions of Software Engineering*, 25(4):557–572.

Shalloway, A., Beaver, G., and Trott, J. R. (2009). *Lean-agile software development: Achieving enterprise agility*. Pearson Education, Inc. / Addison-Wesley, Boston, Massachusetts, USA.

Shenhar, A. J., Dvir, D., Levy, O., and Maltz, A. C. (2001). Project success: A multidimensional strategic concept. *Long Range Planning*, 34(6):699–725.

Shingo, S. (1989). *A study of the Toyota Production System*. Productivity Press, New York, NY, USA.

Shinkle, C. (2009). Applying the Dreyfus model of skill acquisition to the adoption of Kanban systems at software engineering professionals (SEP). In *Agile Conference (AGILE) 2009*, pages 186–191. IEEE Computer Society.

Sillitti, A. and Succi, G. (2006). Requirements engineering for agile methods. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*, pages 309–326. Springer.

Sillitti, A. and Succi, G. (2008). Foundations of agile methods. In Lucia, A. D., Ferrucci, F., and Tortora, G., editors, *Emerging Methods, Technologies, and Process Management in Software Engineering*, pages 249–270. John Wiley & Sons, Inc.

Sommerville, I. (2007). *Software Engineering*. Addison-Wesley, Harlow, UK, 8th edition.

Sutherland, J. (1995). Business object design and implementation workshop. In *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum)*, OOPSLA '95, pages 170–175, New York, NY, USA. ACM.

Sutherland, J., Downey, S., and Granvik, B. (2009). Shock therapy: A bootstrap for hyper-productive scrum. In *Agile Conference (AGILE) 2009*, pages 69–73. IEEE Computer Society.

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

Tata, J. and Prasad, S. (2004). Team self-management, organizational structure, and judgments of team effectiveness. *Journal of Managerial Issues*, 16(2):248–265.

Teasley, S., Covi, L., Krishnan, M. S., and Olson, J. S. (2000). How does radical collocation help a team succeed? In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 339–346, New York, NY, USA. ACM.

Thadani, A. (1984). Factors affecting programmer productivity during application development. *IBM Systems Journal*, 23(1):19–35.

Tichy, W. (2000). Hints for reviewing empirical work in software engineering. *Journal of Empirical Software Engineering*, 5(4):309–312.

Tolvanen, J.-P. (1998). *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence*. Number 47 in Series of Doctoral Theses of Jyväskylä Studies in Computer Science, Economics and Statistics. University of Jyväskylä, Jyväskylä, Finland.

Tuckman, B. W. and Jensen, M. A. C. (1977). Stages of small-group development revisited. *Group and Organizational Management*, 2(4):419–427.

Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *European Journal of Information Systems*, 4(1):74–81.

Watson, T. (2002). *Organising and Managing Work: Organisational, managerial and strategic behaviour in theory and practice*. Prentice Hall, Pearson Education, Harlow, UK.

Watson, T. (2006). *Organising and Managing Work: Organisational, managerial and strategic behaviour in theory and practice*. Prentice Hall, Pearson Education, London, UK, 2nd edition.

Weinberg, G. (1971). *The Psychology of Computer Programming*. Van Nostrand Reinhold, New York, NY, USA.

Weiss, D. M., Bennett, D., Payseur, J. Y., Tendick, P., and Zhang, P. (2002). Goal-oriented software assessment. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 221–231, New York, NY, USA. ACM.

Williams, L. and Cockburn, A. (2003). Agile software development: It's about feedback and change. *IEEE Computer*, 36(6):39–43.

Winter, R. (1994). Self-directed work teams. In *Proceedings of 1994 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop (ASMC)*, pages 123–125. IEEE.

Wohlin, C., Runeson, P., and Höst, M. (2000). *Experimentation in software engineering: An introduction*. Kluwer, Boston, Massachusetts, USA.

Womack, J. P. and Jones, D. T. (2003). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. Simon & Schuster, London, UK.

Womack, J. P. and Jones, D. T. (2005). Lean consumption. *Harvard Business Review*, 83(3):58–68.

Yasin, M. M., Czuchry, A. J., and Alavi, J. (2002). Project management practices: Then and now. *Thunderbird International Business Review*, 44(2):253–262.

Yin, R. (1991). *Case study research: Design and methods*. Sage Publications, Thousand Oaks, CA, USA.

Zmud, R. W. (1980). Management of large software development efforts. *MIS Quarterly*, 4:45–55.