UiT THE ARCTIC UNIVERSITY OF NORWAY

School of Business and Economics

Agile project management

A case study on agile practices

—

André Henriksen

Master's Thesis in Business Administration - February 2016

Acknowledgements

This master thesis concludes my experience based master study in business administration at UiT the Arctic University of Norway.

Working in the software industry as a developer and project manager, I have seen a lot of different approaches to planning and developing software projects. I have worked on both agile and more traditional plan driven projects. In my experience the agile approach seems to be better at creating the end result that the customers want, especially when the customers are unsure what they want.

When I got the chance to immerse myself in a topic, agile project management quickly came to mind. While working on this thesis, I have come to realize that knowing how to code software products is not enough. A lot of other factors plays a role in securing the success of a project. In order to achieve the success every project deserves, the agile approach will help you create the best possible product with the resources you have available. This is the assumption I started out with and wanted to test with this thesis.

I would like to thank my supervisor Sven Arne Pedersen for his guidance, useful comments and insights. I would also like to thank colleagues and friends who gave me input and ideas throughout the writing process. A special thanks to everybody affiliated with Facility Management and WhatIf who participated in the study.

Finally I would like to thank my wife Ingunn and our fantastic children Håvard J. and Martin J. Thank you for your input, support, patience and nudges when I needed it.

Tromsø, February 2016

André Henriksen

Abstract

In project management, and particularly software project management, there has been a shift from traditional plan based project management to the agile event driven project management style. Agile project management has for some time been viewed as the new big change that will revolutionize the software industry.

The Project Management Institute (PMI) was founded in 1969 and offers industry leading certifications for project management professionals. PMI, which was traditionally best known for its traditional project management certifications, have just recently started offering an agile alternative. This new certification (PMI-ACP) has become the fastest growing PMI certification at PMI (Project Management Institute, 2014, 2015).

It can certainly be said that agile is becoming the new standard for software development projects. The concept of agile has been around for some time, and although knowledge and usage are increasing, agile is not always the answer. Understanding when to use agile and which success factors to considered, is important to achieve success.

Agile is still popular and new methodologies keeps appearing. A new popular method, called *The Lean Startup,* was created in 2011. This method advocates that using agile practices is not enough. In order to successfully launch a new company, project or product, there has to be a strong focus on the customer as well (Blank, 2013). Even if the term *agile* was coined quite a few years ago, new agile methodologies are still being created with promise to do agile even better.

The goal of this thesis is to identify possible factors which could affect the success of an agile project. More specifically I want to check which *agile practices* are important to succeed in an agile project. In order to figure this out, a case study among participants in agile projects were conducted.

In this study 53 practices, which could potentially affect project success, were identified through a literature review. These were narrowed down to 23 by asking the respondents to identify practice which were heavily used in their projects. Finally, only 15 of these were found to be especially relevant in agile projects. Six practices were related to *quality*, eight were related to *scope* and one was related to *time*.

The results indicated that practices which improves customer *feedback*, helps the team to understand *customer needs* and improves the team process, are most likely to affect project success.

Keywords: Agile, practices, methods, project management, scrum, extreme programming, XP, Kanban, success factors.

Table of Contents

Table of figures

Tables

Abbreviations

APM	Agile Project Management
PM	Project Manager
PMCL	Project Management Life Cycle
PO	Project Owner
TPM	Traditional Project Management
XP	Extreme Programming
WiP	Work in Progress
TDD	Test-Driven Development
BDD	Behavior-Driven Development
DoD	Definition of Done

1 Introduction

In this chapter I will describe the background and reason for writing this thesis, narrow down the research question and give an overview of the content of this report.

1.1 Background

The theme of this thesis is to search for important factors for achieving project success in an agile software project. The focus will be on agile methods and agile practices.

Agile project management (APM) is becoming the new de facto standard for developing software. More and more companies are using agile to deliver software faster and in a smarter way (VersionOne, 2015). Agile was originally developed for software projects, but because of the potential benefits, it is now used in other types of projects as well (Serrador & Pinto, 2015).

A systematic review of agile software development was done in 2008 (Dybå & Dingsøyr, 2008). The conclusion was that, although a lot of research has been done on agile software development and several benefits and limitations have been identified, the strength of this evidence is very low. According to the review, more high quality studies in agile software development are needed. As such, this thesis is therefore relevant in order to achieve this.

1.1.1 Software project history

Software projects date back as far as the late 1960s. The software industry grew fast and computer companies saw the potential in software production, which had a low cost compared to hardware production and circuitry which were more common. Software companies adopted the already well known *waterfall* model for its software projects. It turned out that this linear approach for developing software was less than optimal (Mens, 2008). The inflexible separation of phases and the fact that requirements are not always clear at the start of a project, were two major limitations of this model. The main causes for software project failures were; *unrealistic project goals, poor estimates, badly defined requirements, poor status reporting, unmanaged risk, poor communication, use of immature technology, high project complexity, poor development practices, poor management, stakeholder politics* and *commercial pressures* (Charette, 2005).

A new approach was needed and several new *lightweight* methods started appearing in the late 1980s and throughout the 1990s. These methods advocated an *iterative* and *incremental* approach to development. This was meant to facilitate a closer collaboration with the customer by encouraging changes throughout the project, in order to better support the customer needs.

In 2001, representatives from several of the most important lightweight methodologies met to discuss and find common ground. They formed the *Agile Alliance*, and the *Manifesto for Agile Software Development* (Agile Alliance) was created. They defined four agile values which all agile methodologies must conform to. These four values were further backed by 12 principles. See appendix B.

The four agile values are:

- Individuals and interactions, over processes and tools
- Working software, over comprehensive documentation
- Customer collaboration, over contract negotiation
- Responding to change, over following a plan

Although the elements to the right are important, the elements to the left are *more* important. This is a clear contrast to the traditional plan-driven approach that TPM advocates. Adhering to these values and principles are meant to increase the likelihood of success.

1.1.2 Project success rates and agile usage

The number of failing or contested software projects each year is high. In a recent CHAOS Manifesto (The standish Group, 2013) it is reported that only 39 percent of all projects can be classified as a success. 43 percent were contested (late, over budget and/or missing features) and 18 percent failed (cancelled or delivered but never used). There has been a gradual increase in success rates, up from 29 percent in 2004.

The report further indicates that project size is more important than whether or not the methodology used is agile or traditional. They argue that the reason for this is that agile methodologies makes is easier to create small project, and a large project is 10 times more likely to fail compared to a small project.

Agile knowledge and agile usage is increasing. The 9th State of Agile Survey (VersionOne, 2015) indicates that 45 percent of respondents are using agile in most of their projects. 90 percent of the organizations in the survey report *some* usage of agile development. *Scrum* is the preferred methodology, used by 56 percent of agile teams. If the various Scrum hybrids are included, this number grows to 72 percent.

1.2 Research question

A common view in the software industry is that by using an APM style, a software project is more likely to succeed. Even though the strength of the empirical evidence is low, agile has for some time been perceived as something that will revolutionize software development and software projects (Dybå & Dingsøyr, 2008). A recent study on agile project success suggests that there might be some truth in this. Serrador and Pinto (2015) conducted a large-scale quantitative study to test if using agile methods has an effect on project success. They found indications that the use of agile methods correlated to a higher reported success rate. This was shown for three categories of success; *overall project success*, *efficiency* and *stakeholder success* (Serrador & Pinto, 2015).

The research in this thesis will be on success in agile projects with a focus on which agile practices are considered most important in order to achieve project success.

The following research question is defined: **How can agile practices contribute to increased project success in small software projects?**

Agile projects are characterized as having a more flexible process compared to traditional projects. This process is made of a set of *practices*, which describes the routines the project team are using in order to achieve the project goals. For *software projects*, which are projects where the goal is to create a working software product, this agile approach assists in defining the project scope throughout its lifetime. Scope, as well as time, cost and quality, are important criteria when considering the *success* of a software project.

1.3 Report structure

In chapter 0, *Introduction*, I gave some background information on project management and briefly looked into the transition from traditional to APM. I also gave a brief definition of software projects. Some metrics on agile success rates were also added. Finally I defined the research question.

In chapter 2, *Case Study,* I will describe the companies participating in this study as well as the projects the respondents used when answering questions.

In chapter 3, *Literature review,* I will describe project management with a focus on APM. I will also look into previous research on agile success factors, describe some of the most popular agile methods and look into some commonly used agile practices. Finally I will add a conclusion and link the literature review to the research question.

In chapter 4 *Methodology,* I will explain the research methodology I have used, define and explain variables, and discuss the validity and reliability of the study.

In chapter 5, A*nalysis,* I will perform an analysis of the case study, look for important agile practices and see if the findings contributes to answering the research question.

In chapter 6 *Conclusion,* I will present the conclusion and discuss limitations and future work.

2 Case study

In this chapter I will present the companies participating in the study and describe the project each respondent had in mind during the interview. Information was gathered from their website and the interviews.

2.1 Facility Management AS

The company

Facility Management (Facilit) (www.facilit.com) is a small software company located in Tromsø, Norway. They specialize in online facility management systems (FMS). Facilit was founded in 1999. Today they are nine employees. In addition they have six people in Colombo, Sri-Lanka, working for the company through an outsourcing agreement.

FMS are used for managing, operating and maintaining buildings and properties. Facilit's main product is called *Facilit FDVU* and customers range from small to large organizations in both private and governmental sectors. Facilit is considered a leading player in the FMS marked in Norway.

The project

Their main solution, Facilit FDVU, was launched in 2009 as a web based FMS solution. In 2012 they realized that mobile support was less than optimal. In February 2013, Facilit made an agreement with an outsourcing company located in Sri-Lanka. A four-man team was formed. The goal was to enhance the existing solution with mobile apps for Android and iPhone. They decided to use the Scrum framework.

The project manager (PM) had no previous experience with Scrum or Agile, but the rest of the team had worked in several agile teams earlier. The PM is located in Tromsø. The rest of the team are located in Sri-Lanka. The project has been running for almost three years.

Scrum was selected because the end solution was not defined yet and they needed a flexible approach for planning and implementing it. Scrum also had the added benefit of *sprints*, which shielded the development team from interruptions. The team could focus on items in the *sprint* without having to worry about changing priorities and other interruptions. Being a Scrum team they try to adhere to the Scum rules and most Scrum practices are therefore used. They are also using several practices from the agile method Extreme Programming.

2.2 WhatIf AS

The company

WhatIf (www.whatif.as) is a software company located in Harstad, Norway. WhatIf was founded in 1997 and was originally a consulting company. They now specialize in applications for doing risk analysis in organizations. Their first product, called WhatIf, was released in 2007. In 2011 the product was released for web browsers. WhatIf has four employees in Harstad. They also have four additional people working for them through an outsourcing agreement. All development is done through outsourcing.

Risk analysis software, as defined by WhatIf, is software which aims to log, reduce and manage risk in an organization, as well as predict possible future risk. The software is made to be simple, so that everybody in an organization can contribute to the risk analysis.

The project

Before 2011, the WhatIf solution was based on a physical board with different *cards* which represented various types of risk. In order to create a digital version of this board, a software projects was formed together with an outsourcing company in Asia. They released the first version in 2011. The project has been running for four years.

The PM had little experience with APM, but the rest of the team had worked in agile projects before. The PM sits in Harstad, while all the developers are located in Asia.

The first few years they used Scrum. The process was modified over time, and now, four years later, they are using a custom method. They started out with Scrum because they liked the idea of a *sprint* based method, since it made it possible for the team to select a number of item each sprint and expect to be able to deliver them. About three years into the project, they switched to Kanban, because priorities started to change rapidly, and the fixed *time boxed* sprints no longer worked for them. They used Kanban for about seven months, after which they started to make additional changes to the process. An ever increasing flow of changing priorities made them look for ways to be even more flexible and more respondent to change. Now they use a custom process, which is heavily based on lean principles. R*apid prototyping* is their principal way of planning and creating software. Their way of doing this is basically by creating a mockup of new features, presenting it to the end users to get feedback, and finally develop it if the feature is relevant for the users.

Because of this graduate change from Scrum to the custom lean based approach, determining which practices they actually use is not that straight forward. Participants in this projects had a rather different understanding of which practices they use, why they use them and how important they is.

3 Literature review

"Project management is the application of knowledge, skills, tools and techniques to project activities to meet the project requirements" (PMBOK 2013: 47)

In this chapter I will first describe how *success* is defined in this thesis. I will also describe project management in general and particularly APM. Furthermore I will give an overview of some of the most important agile methods and practices, and describe some of the existing research that is done on agile success factors. Finally I will add a summary, linking the literature review to the research question.

3.1 Software project success definition

There are several ways to define the success of a project. The traditional approach is to use the *project management triangle* where *scope time* and *cost* each form a line in a triangle. Often *quality* is included as a separate element (iron triangle).

Scope is the set of functional elements (features) delivered during the project lifetime. *Time* is the actual time required to complete the project with its scope. *Cost* is the amount of resources required to complete the project. Making changes to one of the element in the triangle will affect the others (Atkinson, 1999).

Quality is very important when developing software products. Quality can be defined in several ways, but for software products, there are often three types of quality; *functional, structural* and *process* quality (Chappell, 2013).

1. *Functional quality* refers to how well the product works for the intended user. Specifically, how well the software conforms to the defined requirements and design, how many bugs there are, how good performance it has and how easy it is to learn and use.
2. *Structural quality* refers to the product's source code quality. This includes robustness, maintainability, testability, efficiency, security and how it conforms to defined coding practices.

3. *Process quality* refers to how the system is created and the process around it. The main attributes of process quality are meeting delivery dates, meeting budgets and a repeatable development process which reliably delivers quality software.

In order for a project to be considered successful, it must be delivered on time, within budget and with all the required features and functions. However, it is not always easy to determine if a particular project is a success or not. Different people (stakeholders) involved in a project might have different views on what constitutes success. While external stakeholders usually looks at *cost* and *time*, internal stakeholders often use *scope* and *quality* as the most important criteria for determining success (Agarwal & Rathod, 2006; Bryde & Robinson, 2005).

Wicks and Roethlein (2009) defines quality as "the summation of the affective evaluations by each customer of each attitude object that creates customer satisfaction". As an important part of *functional quality,* they consider *customer satisfaction* the most important part of quality. This is also supported by the 9th annual State of Agile survey (VersionOne, 2015) where *customer/user satisfaction* was ranked third on how success is measured. *On-time delivery* and *product quality* were ranked first and second respectively. *Business value* and *product scope* were ranked fourth and fifth.

3.2 Project management

A project is a temporary activity which should result in a unique product, service or result. A project has a clearly defined beginning and end. The end state is reached when the project goals are met or the project is terminated for other reasons. Project management is defined as "the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements" (Project Management Institute, 2013). According to the *PMBOK Guide,* project management consists of five process groups*; Initiation, Planning, Executing, Monitoring & controlling* and *Closing* (Project Management Institute, 2013). In traditional project management (TPM) these are done in a linear and incremental fashion. In APM these are done in a more iterative and adaptive way (Wysocki, 2014).

Based on these process groups, Wysocki (2014) defines five different *Project Management Life Cycle* (PMLC) process models. The process models are *linear, incremental, iterative, adaptive* and *extreme/emertxe.* Deciding on which PMLC model to choose, depends on the degree of solution and goal uncertainty. *Linear* and *incremental* PMLC falls under TPM.

They have a low level of solution uncertainty and the project goal is clear. *Iterative* and *adaptive* PMLC falls under APM. They have a high level of solution uncertainty, but the project goal is still clear. *Extreme* and *emertxe* PMLC falls under extreme project management. *Extreme* PMCL have a high level of solution uncertainty and the project goal is unclear. *Emertxe* PMLC is Extreme in reverse. Here the solution uncertainty is low, but the project goal is unclear. Figure 1 illustrates this.

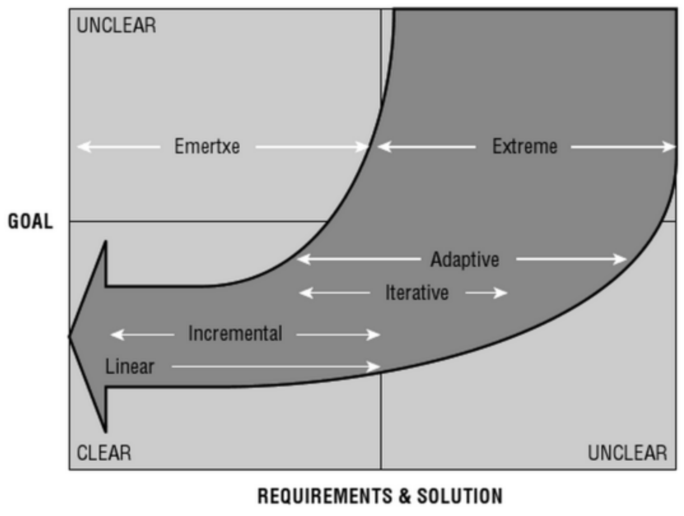

Figure 1 PMLC models

Traditional PMLC

In the *linear* model, each process group is executed exactly once, in sequence. This model does not encourage scope changes. The product is delivered as defined by the scope, which was created at the start of the project.

The *incremental* model is similar to the *linear* model, but here part of the defined scope is delivered in *increments*. This allows for better customer feedback and scope changes are encouraged. The benefit, compared to the linear model, is that the end product will be closer to what the customer wants.

3.3 Agile project management

When the goal is clear, but the solution and how to get there is unclear, an APM model should be used (Wysocki, 2014). These types of projects are defined as *complex* and require a non-traditional approach for a successful project execution.

In TPM everything is planned out into details up front. This is called a plan-driven approach.

In contrast, APM projects are change driven. This means that instead of avoiding changes in the project, changes are encouraged. This causes a more dynamic relationship between the team that creates the product, and the external *stakeholders* (anyone with a vested interest in the product) which requested it. Stakeholders are more involved in the process, but the end result will be much closer to what they really wanted. Since the end solution (scope) is unclear at the beginning, it is defined throughout the project, based on feedback from the customer and other stakeholders.

Agile PMLC

The *iterative* model improves on the incremental model by including planning in the implementation loop. Each loop creates *potential shippable code* (can be launched to production) which the stakeholders can give feedback on. Feedback is welcome and considered an integral part of the process. Based on the feedback and the overall vision of the product, a new *iteration* is planned, executed and deployed. This way, the solution is not fully known at the beginning, but is defined throughout the project lifetime. Figure 2 (Wysocki, 2014) shows this *iterative* model and how most process groups, including planning, are repeated in a loop.

Agile methods are both *incremental* and *iterative*. Incremental because the work is pre-divided into smaller batches of work, and iterative because the scope of each batch is defined just before the start of each loop. This iterative nature makes the process very flexible.

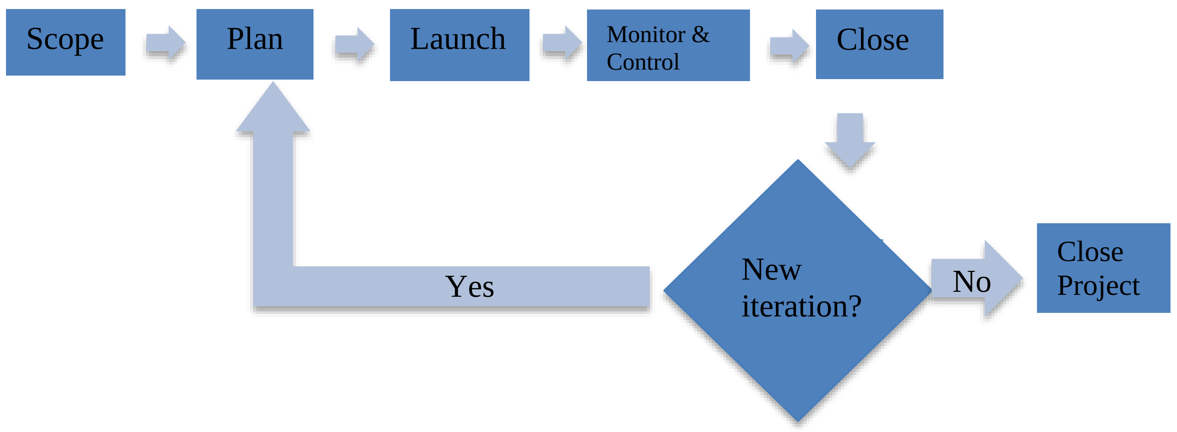

Figure 2 Iterative PMLC model

APM also defined the *adaptive* model, which is very similar to the iterative model, but it has an even shorter loop time, making it easier to respond to changing requirements. The main difference between the adaptive and the iterative model is that more of the solution is unknown in the adaptive model. The less that is known, the more risk and complexity there will be. When complexity is high, an adaptive model is more favorable (Wysocki, 2014).

3.4 Success factors

First in this chapter I will summarize three articles on important success factors in order to get a broad overview of which types of success factors to consider. Second I will focus on practices and summarize two articles which have identified several important *agile* practices.

3.4.1 Agile success factors

A *success factor* is defined as something that "must go well to ensure success" (Boynton & Zmud, 1984). In this context, success relates to the outcome of a software project.

Lindvall et al. (2002) conducted an online workshop with eighteen agile experts from around the world. One of the goals of this workshop was to identify agile success factors. The three most important success factors they found were; *culture*, *people* and *communication.*

Chow and Cao (2008) conducted a survey among agile professionals from 25 countries. Three *critical* success factors and three *possible* success factors were identified; *delivery strategy, agile software engineering techniques, team capability, project management process, team environment* and *customer involvement.*

Misra et.al. (2009) did a large-scale empirical study in order to identify agile success factors. They found nine success factors which were shown to be statistically significant; *Customer satisfaction, customer collaboration, customer commitment, decision time, corporate culture, control, personal characteristics, societal culture,* and *training and learning.*

These three articles indicates a lot of success factors, but there is also a lot of overlap. All three articles mentions factors which could be organized into a grouped called *practices. Communication, engineering techniques, project management process, people and team, customer involvement and satisfaction,* and *decision time* are all examples of identified success factors which is likely to be affected by which practices is being used in a project. Next I will focus on practices as potential success factors.

3.4.2 Agile practices

There are several areas which could be important to achieve project success. Here I will narrow this area down by focusing on *agile practices*.

William et.al. (Williams, Brown, Meltzer, & Nagappan, 2011) did a case study on three Microsoft Scrum teams. They were able to show that by combining an agile framework (Scrum) and nine additional practices, the teams were able to improve quality, productivity and estimation accuracy. In addition to the basic Scrum practices, these nine practices were used:

- Peer review
- Definition of done
- Planning poker (agile games)
- Continuous integration
- Source control
- Static analytic tools
- XML documentation
- Code coverage
- Test-drive development

A systematic review on agile practices was conducted by Jalali et.al. (Jalali & Wohlin, 2012) to determine the status of combining agility with engineering practices. 81 peer-reviews articles were identified, from which 61 were empirical studies. 53 of these described a successful agile implementation and were finally included in the study. 25 agile practices were identified:

- Standup
- Sprint planning
- Sprint review
- Onsite customer
- Coding standards
- Incremental design
- User stores
- Sit together
- Sprints/iterations
- Retrospective
- Test-driven development
- Backlog
- Refactoring
- Automated testing
- Architecture focus
- Enough documentation
- Continuous integration
- Pair programming
- Scrum of scrums
- Integration testing
- Planning games
- Scrum master
- Burn down chart
- Collective code ownership

As shown, there are a lot of practices available. Each practice is described in appendix F. More practices are identified in later chapters.

3.5 Agile methods

"Like all of the agile processes, Scrum is an iterative and incremental approach to software development" (Cohn, 2010b)

Different methods have different practices. When choosing a method for a given project, these practices should be considered in order to make sure they fit with the project. Not all methods describes a lot of practices, but rather focuses on other aspects. Scrum describes practices in form of *events*, *roles* and *artifacts*. Extreme programming describes 24 practices in form of *engineering practices*. Kanban describes five *principles* which also could be viewed as practices. A method can be viewed as a collection of best practices, values and/or principles, which has been proven to work for certain types of projects. In this chapter I will take a closer look at some of the commonly used agile methods.

3.5.1 Scrum

Scrum is an *iterative* and *incremental* framework, specifically created for managing software projects in an agile way. Scrum was defined around 1995, but really increased in popularity after 2001 when the Agile Alliance was founded. The Scrum framework is defined around a set or *roles*, *events* and *artifacts* (Griffiths, 2012). A list of Scrum practices are listed in appendix E.

The Scrum framework only defines three roles; *Product Owner, Scrum Master* and *Team Member.* The Product Owner controls the product *backlog* and defines requirements. The Scrum Master makes sure that the Scrum process is followed and resolves impediments. Team members are other members. I.e. developers, testers, designers etc.

Scrum has four events; *Sprint planning, daily standup, sprint review* and *sprint retrospective.* Each event, which is time boxed (time limited), is there to facilitate quick progress and constant flow of information between all team members and external stakeholders. A sprint is (normally) a 2-4 weekly cycle where a team works on an agreed upon set of tasks.

The most important artifacts are; *Product backlog, sprint backlog* and *product increment.* The product backlog contains requirements to be developed in a future sprint. The sprint backlog contains requirements which are currently being worked on as part of the ongoing sprint. The product increment is the result of the work which was done during a sprint.

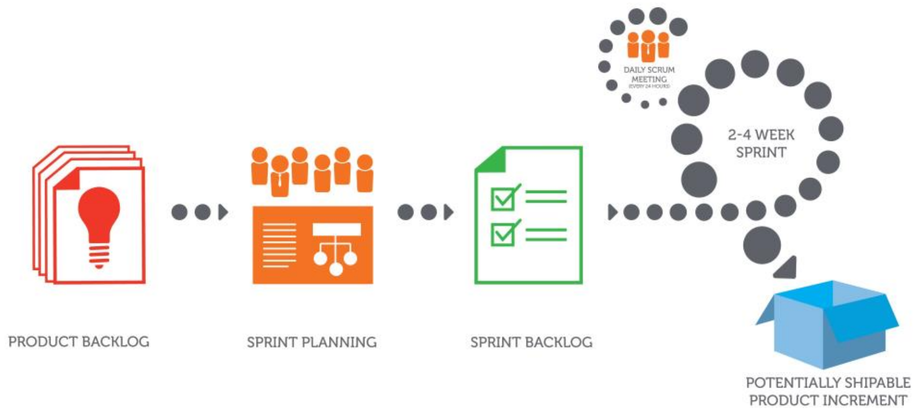

Figure 3 Scrum process flow

In Figure 3 the Scrum process flow is shown. In each sprint, during the *sprint planning*, the *Product Owner* and the *Team* decides on a subset of items from the *product backlog* to put into the *sprint backlog*. The team work during the *sprint* (a time boxed event) to create a *potentially shippable product increment* based on the items in the *sprint backlog*. Every day they have a *daily scrum meeting* in order to share information and solve potential problems.

3.5.2 Kanban

Kanban is more of a mindset than a methodology. This means that it does not offer a lot of specific rules on how to follow a process, but rather focuses on having the correct attitude (Poppendieck & Poppendieck, 2013).

Kanban has a few core principles, and Griffiths (2012) identifies five principles meant to create a lean behavior in a project; 1) *Visualize workflow*, 2) *Limit Work-in-Progress (WiP),* 3) *Manage flow*, 4) *Make process explicit* and 5) *Improve collaboratively*. See appendix C for details about these principles.

Each item selected for development resides in a prioritized backlog called a Kanban board. A Kanban board has multiple columns where each column represent a status. Each item resides in one of the columns, and flows from left to right as it is being worked on. The WiP limit is set for each column, indicating how many items a column can hold at any given time. If too many items are clustered in a single column, it is the responsibility of the whole team to solve the problems and make the flow work again.

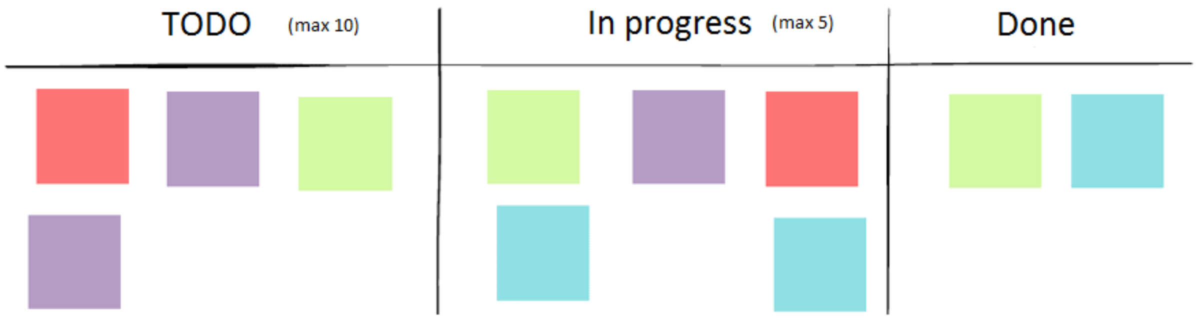

Figure 4 Kanban board example

Figure 4 shows an example of a simple Kanban board. Items placed higher in a column, has a higher priority. This makes Kanban a good choice for maintenance projects where priorities changes frequently.

3.5.3 Extreme programming

Extreme programming (XP) is a software-centric agile method. Kent Beck, one of the original signers of the Agile Manifesto, created XP in 1999 (Beck, 2000). Today XP defines five core values, 15 principles, 13 primary practices and 11 corollary practices (Beck & Andres, 2004). These are meant to improve code quality and product value, and create good development practices (Beck & Andres, 2004; Griffiths, 2012). XP advocates short development cycles (iterations) and really focuses on putting as many practices into action as possible. Primary practices can be adopted quickly and should give instant results. Corollary practices are more complex and requires more experience to implement in a project (Beck & Andres, 2004). See appendix D for more information about the values, principles and practices of XP.

3.5.4 Scrum hybrids

Several variations of Scrum exist. Two of the most popular are *Scrumban* and *Scrum/XP*.

Scrumban is a model based on Scrum and Kanban. Scrumban combines the roles, events and artifacts of Scrum and the Kanban board with its work-in-progress limitation. The big difference is that Scrumban does not use the time-boxing from Scrum. Instead of sprints, the workflow is continuous.

Scrum/XP is a combination of practices and rules of Scrum and XP. This method provides an environment where a customer can evolve a software product into what best meet with the business criteria (Mar & Schwaber, 2002). Five shared practices help facilitate this; *iterations, increments, emergence, self-organization* and *collaboration.*

3.6 Agile practices

In this chapter a synopsis of the latest *State of Agile Survey* is added to give an account of practices currently being used by agile practitioners. Practices especially important in agile projects are also listed and described.

3.6.1 State of Agile survey

The State of Agile Survey, by VersionOne, is an annual survey where thousans of respondens participate each year. It has been running for 10 years, which makes it «the largest and longest running agile survey in the world» [2]. There are a lot of agile practices available and some are more popular than others. In the latest State of Agile Survey (VersionOne, 2015), a list of the 25 most commonly used practices were identified. The percentage in the list below, shows the proportion of respondents who used the specified practice.

The following practices were identified:

- Daily standup (80%)
- Short iterations (79%)
- Prioritized backlogs (79%)
- Iteration planning (71%)
- Retrospective (69%)
- Release planning (65%)
- Unit testing (65%)
- Team-based estimation (56%)
- Iteration reviews (53%)
- Task board (53%)
- Continuous integration (50%)
- Dedicated product owner (48%)
- Single team (46%)
- Coding standards (43%)
- Open work area (38%)
- Refactoring (36%)
- Test-Driven development, TDD (34%)
- Kanban board (31%)
- Story mapping (29%)
- Collective code ownership (27%)
- Automated acceptance testing (24%)
- Continuous deployment (24%)
- Pair programming (21%)
- Agile games (13%)
- Behavior-driven development (9%)

There is a lot of overlap between this list and the list of practice identified in chapter 3.4.2. They are all described in appendix F.

3.6.2 Agile engineering practices

As shown in earlier chapters, there are a lot of practices available. However, most are also used in traditional projects. I will here describe practices especially relevant to agile projects and therefore is relevant when searching for practices which are likely to affect success in agile projects. The words *sprint* and *iteration* are used interchangeably and both should be understood as a *development cycle.*

Burn down chart

A *burndown chart* is a graph which shows how much work is remaining in an iteration (or in the entire project). It also shows how much time has passed since the start of the iteration/project. This practice gives everybody involved an up-to-date view of the iteration/project status. This transparency is also meant to encourage the team [1].

Daily standup

Daily standup is an event where the development team meets every day for a few minutes to coordinate development and share important information. Each member describes *completed* work and any *impediments* they need resolved before continuing. The meeting should be short and to the point [1].

Definition of Done

Definition of Done is a practice where the team decides on a set of criteria which must be satisfied in order for an item/story to be considered complete. Everyone in the team should have the same understanding of what it means when an item/story is considered *done* [1].

Iterative development

Iterative development is the practice of including planning in each development cycle. I.e. deciding what the scope for the next cycle is [1]. One of the benefits is that you do not waste time building the wrong product. As new features are built, feedback from customers help steer the product in the right direction, and scope is defined and redefined throughout the project.

Incremental design

The essence *of incremental design*, is that, instead of designing the whole system at the start of a project, when you might not know exactly what the product should look like, you only design what you need with the information you have available [1].

Product backlog

A *product backlog* is a list of features and technical work which are likely to be necessary in the current project. It should be prioritized with high priority items on top. This is the full scope of yet to be developed features, but it is subject to change over time [1] [4].

Product owner

The *product owner* (PO) is a Scrum defined role. The PO should preferably be a *single dedicated* individual, who is responsible for managing the *backlog* and making sure items selected for development maximizes the value of the product [4]. Only the PO is allowed to assign work to the team and make changes in the backlog. A PO is not the same as a project manager. The PO does not manage the team, he only manages the backlog.

Scrum Master

The *scrum master* is another role defined in Scrum. The scrum master is a servant leader whose main responsibilities is to make sure the team adheres to the scrum theory, practices and rules. The scrum master should help the team to maximize their production value in any way he can [4].

Sprint planning

Sprint planning (or iteration planning) is a Scrum event where the team meets to discuss and decide what they will work on in the next iteration. The PO (if one exists in the team) presents the objective of the upcoming iteration, and explains which items from the backlog should be included in order to achieve the sprint objective. The development team selects items they can commit to and forms, together with the PO, the goal for the sprint/iteration [4].

Sprint backlog

The sprint backlog is a Scrum defined artifact, which contains a list of items the development team think they can complete in the current sprint (iteration). It is defined during *sprint* *planning* but can be changed during the sprint by the development team. Every item in the sprint backlog should be well described and well understood by the team [4].

Sprint retrospective

Sprint retrospective is a Scrum defined event, where the team meets after a sprint to discuss what worked well and how they can improve. During this event they inspect the last sprint and look for ways to improve the next one [4]. The goal is to make the team better.

Sprint review

The *sprint review* is another Scrum event where selected stakeholders meet with the development team in order to inspect changes and deliver feedback. It is not a status meeting, but rather an informal meeting where the goal is to get feedback, foster collaboration and discuss future work [4].

Team member

The *team member* practice is a Scrum practice which states that, the only allowed role in a Scrum team, apart from the Product owner and the Scrum Master, are *team members.* Developers, testers, graphic designers etc. are all considered team members and has no other role in the team. This model is designed to foster flexibility, creativity and productivity [1].

Stories

A *story*, often called a *user story,* is a user centric way of describing a new feature. A story is written in a certain style in order to highlight *who* needs the feature, *what* they need and *why* they need it [1].

Task board

A *task board* is a board with status columns, where each item/story is placed based on its status. This is a common way to show a team's progress as well as the status of every item [1]. Tasks boards can be digital or physical. On physical task boars a story is often just written on a post-it note and moved from column to column as it transitions from *start* to *done*.

Visualize Workflow

Visualize workflow is a Kanban practice which says that it should be easy to see how items are transitioning form start to finish. This is normally done with a task board. In Kanban the task board is called a Kanban board, and it has a work-in-progress property on each column (Hammarberg & Sundén, 2014).

Limit work-in-progress

The work-in-progress (WIP) limits in a Kanban board are meant to prevent road blocks and make items flow faster through the board (Hammarberg & Sundén, 2014). The idea is that if a column contains more items than the WIP limit, it is the responsibility of the whole team to assist and make sure the *block* is resolved. This WIP limit is what distinguishes a *normal* task board form a Kanban board.

Improve collaboratively

Improve collaboratively is a practice which encourages team members to improve the team and its process continuously through small changes (Hammarberg & Sundén, 2014). The goal of this practice is somewhat similar to the goals of the *sprint retrospective* practice, but the focus is more on improving the collaboration process and the way people work together.

Team based estimation

Team based estimation is a practice where all team members, at least those involved in coding, takes part in estimating a task. It takes more time when the whole team is involved, but the result is a more accurate estimation.

Agile games

Agile games is a common name for all game-like practices which are meant to make part of the process in agile teams easier, funnier or better. One well known example of an agile games is *planning poker* which is a way to estimate user stories more accurately.

Whole team

Agile teams are often said to be *self-organizing* and *cross-functional*. Self-organizing means the team itself is responsible for deciding how project challenges are solved, based on boundaries and constraints set by management (Cohn, 2010a). Cross-functional, also called *whole team,* means that the team should possess all the knowledge and skills to create the feature they have been set to create (Beck & Andres, 2004). Creating a team of experts is likely to affect the cost of the project as well, because experts are generally more expensive than novices. However, if will also reduce the amount of time needed to complete a project. Barry Boehm’s principle “use better and fewer people” is an important principle for agile projects (Cohn, 2003).

53 practices were identified in chapter 3, and 21 of these were found to be especially relevant in agile software projects.

3.7 Summary

In this chapter I looked at several topics in order to create a theoretical foundation for this thesis. In chapter 3.1, I described how project success is defined for software projects by looking at the four elements; *scope*, *time*, *cost* and *quality*.

In chapter 3.2 and 3.3, I gave some background information on traditional and APM.

In chapter 3.4, several success factors were identified through a literature review. Three articles on *success factors* in agile projects where described. Focusing on *practices*, I also described two relevant article on *agile practices*.

It was shown in chapter 3.5 that there are a lot of different ways to be agile. Different agile methods have different practices, tools and techniques. There are a lot of practices, and they are probably not equally important to achieve success.

In chapter 3.6, I summarized the 25 most heavily used agile practices as identified by the State of Agile Survey. Combining this with all the identified practices from chapter 3.4, I finally described practices especially important for agile projects.

Based on information gathered this chapter 3, it is clear that there are a lot of potential success factors in agile software projects. I have decided to limit my research to the agile *practices* and try to figure out which practices are most important in order to achieve project success.

4 Methodology

In the methodology chapter, I will describe the process which resulted in the given research question, I will also give an overview of which choices were made when selecting the research design and why. Finally I will explain the selection criteria and look at the validity and reliability of the study.

4.1 Research question and research design

In chapter 1.2 the research question was defined as*: How can agile practices contribute to increased project success in small software projects?* I will here explain how I decided on this particular question, and which limitations I have set.

4.1.1 Pilot study

After creating the initial interview guide, I conducted a pilot study with an agile professional. This interview was done in order to see whether the respondent would agree with the agile values, principles and success factors I had already identified in chapter 3.

An interesting point was made by the respondent. He felt that engineering practices are given to little attention in agile trainings. This is a major flaw, in his opinion, because it is the practices which makes a project successful. Going fast is not enough, you need these engineering practices to also enhance quality.

The pilot study helped me to focus my research on agile practices as potential success factors.

4.1.2 Research question

When I started working on this thesis, the plan was to search for all factors that might affect the likelihood of project success. After discussing it with my supervisor and conducting the pilot study, it became apparent that this approach was too comprehensive. A new narrower question was defined.

In this new question, the goal is still to look for factors which increase the chance of project success, but the focus will be on agile *practices*. The goal is to see which agile practices are being used, as well as why and which effect they have.

A research question should conform to three requirements (Jacobsen, 2005). It should be *exiting*, in the sense that the result of the research should not be known in advance. It should be *simple*, in the sense that the research question should be narrow and therefore easier to manage. It should be *fruitful*, meaning that it should be possible to research empirically and it should add something to the existing knowledge base. I feel that the research question I have selected adheres to these requirements and is therefore a good question.

4.1.3 Research design

For the research design I opted for a *holistic multiple-case descriptive case-study*. This was achieved through an *open-ended interview with multiple respondents*. In order to make sure the candidates did not forget any practices, a short quantitative multiple choice survey was also included as part of the interview. I will explain the rationale behind each element in this design.

In a *holistic* design, the study involves units of analysis at only one level (Yin, 2014). A holistic design was selected because of the limited time and limited number of available candidates. In each case I will therefore only focus on one project.

Two drawbacks with a holistic design are; 1) an increased risk that the study is done on a too abstract level, causing the data gathered to lack depth, and 2) an increased risk that the nature of the study may change during the study, and unless the design is changed appropriately, the study can be conducted with the wrong design (Yin, 2014).

The Advantage of using a *multiple-case* design, compared to a single-case design, is that the evidence is often considered more robust and compelling. The main disadvantage is that it requires more time and resources. By looking at multiple cases, it is possible to achieve a degree of *literal replication* (Yin, 2014). If the result from the study show similar results between the respondents, some generalization can be claimed. More cases with the same reported success factors increases the likelihood of it being a general factor for success. However, generalization is not the goal of this thesis.

Yin (2014) distinguishes between *explanatory, exploratory* & *descriptive* case studies, and he describes a *descriptive case study* as “a case study whose purpose is to describe a phenomenon in its real-world context”. In this study, where I am looking at specific practices (the phenomenon) and trying to research them in-depth, a descriptive case study was selected.

Furthermore Yin shows that regardless of which design is selected, a case study can be the appropriate research method. Whether a case-study should be selected or not, depends on the research question. If the research question is of a *how* or *why* nature, and the focus is on a contemporary event, the case-study is one of the appropriate methods. In addition, because of the limited access to relevant respondents, I have opted for a case-study.

The qualitative approach with individual interviews is a good way to get a lot of in-depth information from each case. With the open-ended approach, the respondents can somewhat affect the direction of the interview and this allows for a deeper understanding of the respondents thoughts on the subject.

Variables

The dependent variable in the case study is defined as *project success*. *Practices* is the only defined independent variable.

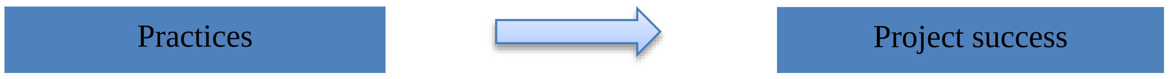

Figure 5 Research variables

Figure 5 illustrates how the independent variable *practices* affecting the dependent variable *project success*.

4.2 Case study

Here I will describe the selection criteria and the respondents who participated in the case study. I will also describe how this case study was implemented and explain how the interview guide is structured.

4.2.1 Selection criteria

In this case study I wanted to interview team members from agile projects. In order to get as much details as possible, two respondents from each project was necessary. The goal was to determine which practices they used and how (if at all) these would affect the likelihood of project success.

The respondents of this study are largely selected because of convenience. Respondents were selected from people I know and who were willing to participate in the study. They do therefore not necessarily represent the best possible choices to enlighten the research question. However, efforts were made to try to include people with various background and experience.

4.2.2 Respondents

Respondents from two projects participated. From each project I recruited two respondents, the PM and one developer. The projects are similar in several ways. They are run in small IT-companies which create and maintain software solutions. The teams are small with a handful of people using an agile approach. Both companies outsources part of their core business. The company and their projects were described in chapter 2.

Four respondents were interviewed. I will use the designation *PM1* and *PM2* for the project managers, and *D1* and *D2* for the developers in the analysis. I will use the designation *P1* when referring to project 1, and *P2* for project 2.

- PM1 has worked 5 years in this company and in this domain. He has no prior experience with project management. This is his first agile project. He has mainly worked as a developer and has 8 years of experience. He is located in Norway.
- PM2 has worked 5 years in this company. He has several years of traditional PM experience. This is his first agile project. He is located in Norway.
- D1 is a developer with 3 years of experience in this domain. He has 2 years of experience in traditional projects and 4.5 years of agile experience. He is located in Asia.
- D2 is a developer with 4 years of experience in this domain. He has 2.5 years of experience in traditional projects and 4.5 years of experience in agile projects. He is located in Asia.

4.2.3 Implementation

The respondents were interviewed one by one. One respondent was interviewed face to face. Three respondents were interviewed via Skype using a video feed. Two candidates were interviewed in Norwegian while two were interviewed in English. All interviews were recorded (audio only). An open-ended interview style was used on order to avoid interview bias and really capture the actual thoughts of the respondent. The interview guide in Appendix A was used when conducting the interviews.

Before the interview started, I explained the agenda and we used a few minutes to make sure everybody understood what was expected of them. This gave us a smooth transition into the actual interview which was conducted without any problems.

4.2.4 The Interview guide

The interview guide in Appendix A was created based on the research question and research done in the literature review.

The guide has two sections. In the first section, questions related to the company (question 1), candidate (question 2) and case under study (question 3) were defined. This gave me some background information on the company and project, and also serves as a way to let the candidate answer some simple questions first.

The second section contains questions related to the agile practices used in the case under study. First (question 4) there will be an open discussion where the candidate explains how the team worked together based on the agile method and the practices they used. The goal is to get an in-depth account of the process they used. Then (question 5) the candidate completes a quantitative multiple choice survey, indicating which practices they used in the project and whether it was heavily used, somewhat used or not used. This was done in order to make sure the respondents remembered all practices. After grouping these practices into *heavily used, somewhat used* and *not used*, the candidate is asked to elaborate (question 6) on these answers in a qualitative open ended style. A final question (question 7) is added to check if the candidate has anything else he wants to add.

4.3 Validity and reliability

When doing an empirical research study, it is important to determine its quality. In order to determine if the study can be trusted, if it is credible, if it can be confirmed and if the gathered data can be depended, the validity and reliability of the study must be checked. Four *tests* are commonly used in order to achieve this; *construct validity, external validity, internal validity* and *reliability* (Yin, 1994). Based on the details below, my conclusion is that this study is both valid and reliable.

4.3.1 Validity

Construct validity is defined as "establishing correct operational measures for the concepts being studied" (Yin, 1994). Construct validity are ensured because of several factors. The questions used in the interview are created based on a thorough literature review where important agile practices and earlier research on agile success factors were looked at. These questions were also reviewed by the supervisor of this thesis. Because four interviews were conducted, this can constitute multiple *sources of evidence*, an important construct for achieving validity.

Internal validity is defined as "establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships." (Yin, 1994). In order to make sure the results and the conclusions fit with what we would expect them to be, initial results were shown to some of the respondents in the study. None of the results were unexpected.

External validity is defined as "establishing the domain to which a study's findings can be generalized" (Yin, 1994). For case studies, generalizing can only be done through *analytic generalization*. If several cases in a study can be shown to support the same theory, replication can be claimed and generalization is possible (Yin, 1994). It is however not the goal of this study to generalize the result.

4.3.2 Reliability

Reliability is defined as "demonstrating that the operations of a study (…) can be repeated with the same results" (Yin, 1994).

In order to achieve reliability, a case study database was created. All interviews were stored as audio files and later transcribed. The collected data can therefore be made available for other researchers.

To make sure I, as the researcher, did not affect the answers given by the respondents, I made sure I followed the interview guide throughout the interview. This helped me focus on the questions and not steer too far of the planed path. In order to avoid misunderstand, I made sure to explain all the practices we talked about if that was necessary. In cases where it turned out I had asked leading questions, I chose to put les emphasis on those answers in the analysis.

Because the context of the interview can affect the answers given by the respondents, all interviews were made during normal office hours. The respondents sat in their office, or in a meeting room close by, during the interview. No interruptions occurred during the interview.

5 Analysis

In this chapter I will present the empirical data and link it to the theory in chapter 3. The analysis will be grouped by the project management triangle constraints which were described in chapter 3.1.

Practices described in this chapter have been identified as heavily used and will be analyzed to see how important they are. This is based on the quantitative survey and the open ended discussion with the respondents. Several practices were concluded to be heavily used even though some of the respondents initially indicated them as somewhat used or even not used. Furthermore, I will only focus on practices which are especially important in agile projects. Practices which are equally important in traditional projects, will not be analyzed.

A total of 53 practices were looked at and 23 were found to be heavily used. 15 of these were especially relevant in agile projects. Eight of these were related to *scope,* one was related to *time* and six were related to *quality*. The analysis is therefore divided into three groups; s*cope*, *time* and *quality*.

None of the identified agile practices were mainly *cost* related, although some of them can affect cost as well. Cost is therefore not included as a separate group. Several practices could be placed in more than one group, but I have placed them in the group where they fit best, based on how the respondents understood these practices.

5.1 Scope

Scope refers to the set of functional elements (features) delivered during the project. Eight practices related to *scope* were identified as heavily used. *Iterative development, product backlog* and *stories* are heavily used in both projects. *Sprint planning, sprint backlog, sprint review* and *team based estimation* are heavily used in P1. *Incremental design* is heavily used in P2. This is illustrated in Table 1.

Table 1 Heavily used scope related practices

Scope practices	**P1**	**P2**	**Both**
Iterative development	X	X	X
Product backlog	X	X	X
Stories	X	X	X
Sprint planning	X		
Sprint backlog	X		
Sprint review	X		
Team based estimation	X		
Incremental design		X	
Sum	**7**	**4**	**3**

Iterative development

In order to be agile, you must be able to quickly adapt to changing priorities. Working in short iterations gives a team this option. In P1 both respondents ranked this practice with *high* importance if you want to achieve project success. They mentioned several reasons for this. First and foremost they both mentioned *feedback* as the most important reason for doing iterative development. By releasing in small iterations to get early feedback, they are able to create a product which is more in tune with customer needs.

The developer (D1) mentioned three main reasons for doing this practice; *solution uncertainty, time to market,* and *feedback*.

> "*So the requirements for the iterative approach is at the beginning of the project we did not have a 100% clear idea of what we need to do when it comes to features. (...) try to develop (...) in iterations so we can go to market earlier and get the feedback from users as well.*" – D1

When asked directly how this practice contributes to success, the developer (D1) again mentioned stakeholder and user *feedback* as one of the most important ways to achieve success. The project manager (PM1) had a similar idea of why this is a good practice, and especially mentioned feedback and changing priorities.

> "*When things changes rapidly it's nice to be able to change direction and not be locked into a predefined course which lasts for 6-12 months. It is also easier to get feedback*

> *which benefits the product and the quality of the product. We have chosen two week iterations (...) to get feedback as soon as possible if we do something wrong.*” – PM1

He (PM1) also said that it would be difficult to imagine how the project would work without using the iterative development practices. Because they are using Scrum in P1, it makes it easier to adopt this practice. Scrum with its *sprints* is by definition iterative.

None of the respondents in P2 reported this to be a heavily used practice, but they are clearly working in iterations here as well. The main difference is that they do not used fixed time intervals for each cycle, as they do in P1.

> *“It is a real cycle (...) it doesn’t have a fixed time bound.”* – D2

They work more continuously in P2, taking in prioritized changes when needed. By using a *rapid prototyping* practice in P2, they are able to use the iterative approach to quickly respond to *changing priorities*, get early *feedback* and gain a quick *time to market* effect.

> *“We do rapid prototyping. Basically with the requirements, there is a designer in the team who (...) comes up with mockups. And then the designer starts the prototypes (...) the PO uses this prototype to get closer to the customers, to get feedback before starting the implementation. So with this approach, we are starting early and we can also fail early.”*- D2

Even though they have very different approaches on how they do iterative development in P1 and P2, they gain much of the same effects. Working in iterations like this gives the team the ability to create a product which is closer to what the customer wants. It is reasonable to assume that this in turn will increase customer satisfaction, which is likely to affect whether a project is considered successful or not.

Product backlog

Instead of having a large requirement document, agile teams often works from a *product backlog.* A product backlog is heavily used in both projects but they have a different view on how important it is.

Traditional projects might also break a requirement document into a list of tasks, but for agile project using the iterative approach, it is especially important. In P1 the developer (D1) said that there were two main reasons for using this practice.

> "*It sets the overall objective of the product, so we have a long term mission objective (...) also helps breaking this down (...) and have an idea what needs to be taken into the upcoming sprint.*" – D1

The last point is only important if the team works in sprints. The project manager (PM1) also mentions long and short term planning as important reasons for using this practice. In addition he said that having a product backlog, as long as it is prioritized, makes the project more transparent and allows other stakeholders to get an overview of progress and prioritizations.

> "*You need a product backlog (...) the product owner side can see what we have planned, but it is also used to create transparence with the team*" – PM1

Both the developer (D1) and the project manager (P1) made a point of mentioning that this practice is especially important for Scrum, because of the iterative approach. They both ranked this practice with *high* importance if you want to achieve project success.

Even though they have a product backlog in P2, they do not put a lot of emphasis of the importance of it. Their problem is that priorities are changing very rapidly. Because of this, items on top of the backlog may not get picket for development first. This rapid change is the reason why they moved away from Scrum and sprints.

I interpret this to indicate that the *product backlog* practice is important for some agile methods, but not for all. Methods using sprints or other time boxed iterations, would likely benefit from using this practice.

Stories

Items in a backlog can be structured in a lot of ways, but creating *stories* is a popular choice. This practice is heavily used in both projects, but the respondents disagree on how important it is for project success. The main incentive for creating stories were reported by all respondents to be the ability for everybody involved to understand *why* a feature is important for the end user.

> "*It is a nice and short way to (...) get a picture of what the customer needs. It's a good way to make all team members quickly understand what the feature is all about.*" – PM1

> "*I write user stories so they (i.e. the developers) can see what the end result of a task should be, in the eyes of the user. This way they have to understand the user perspective. We feel that this works.*" – PM2

In P1 the developer (D1) also said that this makes it easier to focus on the user's objectives rather than just completing a feature. It affects development in a positive way. He also emphasized that he felt this practice should be used by everybody because "*ultimately the product success depends on whether users get what they expect or not*". The rest of the respondents said that this practice is important to achieve success, but many other practices are more important. Furthermore it seems like this practice is important regardless of which method is used, because creating stories helps the team to understand why and how the user intends to use a particular feature.

Sprint planning

When a team works in iterations, it helps to plan each iteration before it starts. This practice is heavily used in P1, but this is mainly because they are using Scrum. In order to be able to commit to a set of tasks each sprint, they need to plan the sprint first. The project manager (PM1) mentioned that the alternative, planning several months ahead, is a bad idea.

> "*We plan often and for a few features at a time, instead of having huge planning sessions. It causes us to work only with features which are important right now. We plan the most important things at any given time. It is important*." – PM1

Both respondents in P1 indicates this as an important practice to achieve project success because it helps the team to build the right thing. The project manager (P1) said that it was really important while the developer (D1) only found it important because they are using Scrum. He (D1) felt that it was more of an administrative practice that has to be there in order for the Scrum process to work properly.

In P2, which has a more continuous approach, they no longer use this practices, at least not in the same way. When they were using Scrum, this practice was heavily used in P2 as well. They still do planning sessions, but more infrequent and supplemented by regular calls between the PM and individuals in the development team. This infrequent setup works well for them because their process is more continuous and there is no need to plan out the next few weeks regularly. Because they have so many changes in priority, planning too far ahead would just be a “*waste of time*” (PM2).

Sprint backlog

During one of these planning sessions, a *sprint backlog* is often created. This practice is heavily used in P1, mainly because they are using Scrum.

The project manager (P1) mentioned that having sprints and iterations works really well. One reason is that P1 is a development project with little or no running business. I.e. bug fixes and other interferences. There are few change requests, and this makes it easier to use a sprint backlog.

> “W*e have a sprint backlog. It is also prioritized, so we know what should be solved first and last. It’s a good aid to get an overview of each sprint.*” – PM1.

The developer (D1) feels the main effect of doing this practice is to see if the team is succeeding in completing all items in the sprint. The project manager (P1) said that being able to completely focus only on a few items, and not having to consider the whole backlog when deciding what to do next, gives the team more focus and makes communication easier. In addition he likes that it gives him some way to measure progress.

They do not use a sprint backlog in P2. This makes sense since they do not work in time-boxed sprints. They used this practice in P2 when they were using Scrum. This worked for them, and at that time they felt it was an important part of the process.

Using a sprint backlog seems important for Scrum projects, but not for agile methods with a continuous approach.

> “*In a Scrum setting it is very important. But not equally important in other agile methods*.” – PM1

Sprint backlog might be an important practice, but all respondents in P1 agrees that it is not a critical factor for achieving success.

Sprint review

At the end of every sprint/iteration, a *sprint review* is often held. This practice is heavily used in P1. It is a prescribed practice in Scrum, but both respondents in P1 said that this in a really important practice because of the feedback it generates from important stakeholders.

> "*The main advantage was to make sure we demonstrate what we have achieved (...) and to get the feedback. Mainly in order to get feedback."* – D1

> "*Early feedback. To get that feedback loop as short as possible is important. To avoid that the development (i.e. product) moves in the wrong direction.*" – PM1

The project manager (PM1) also mentioned that a lot of people feels there are too many ceremonies (i.e. events) in Scrum, but he feels that this team and this project benefits from all of them. He specially mentions the importance of planning each iteration and reviewing the work and the process after each sprint.

They do not have reviews in P2 anymore. At least not in the common sense. Earlier they included everybody in this meeting, but now it is just the PM and one developer. The PM then informs the rest of the company. They do this because they feel it is more effective. This approach might be more effective, but it is not stimulating feedback in the same way. They compensate the lack of feedback from stakeholders by being in constant contact with customers and checking what they think about new features. The feedback arrives later, but it is probably more accurate because it comes directly from the end users.

This practice is considered important because it stimulates early feedback. My interpretation of this is that having a sprint review, or some other way to stimulate feedback is important for success

Team based estimation

In order to make it easier to know how many items you can expect to complete in an iteration, they can be estimated. This practice is heavily used in P1 in order to get a more accurate estimation. A good estimation will in turn make it easier to plan the iteration.

> "*The effect is that you get a more realistic estimation. This is important because we are using Scrum. (...) we measure velocity based on estimations."* – PM1

> "*It works well, because different team members will have different amount of overview into the code and the complexities involved. We get more accurate estimate for a task.*" – D1

However, it certainly takes more time when everybody in a team have to be present during the estimation process, and this is why they no longer follow this practice in P2. They do not feel that the extra effort of estimating is worth the extra accuracy.

> "*We did it earlier at the start of the project. Now (...) it just takes unnecessary time*" – PM2

Team based estimation does not appear to be the most important practices to use in order to succeed in a project. However, if you need estimations to be more accurate, this is a smart thing to do.

Incremental design

The final scope related practice is heavily used only in P2. The developer (D2) indicated this to be a very important practice for achieving success.

> *"We think ahead and use whatever the design is for now. We keep it flexible for the future."* – D2

The PM in P2 strongly feels that they cannot know what the customer needs 100%. So they plan a feature to about 60%. They then talk with their users and figure out the remaining 40% before they start developing it. This affects the scope of the product and he feels that it makes the product more usable and competitive in the market.

> *"We can't imagine what the customer needs 100% in the future. We ask the customer."* – PM2

In P1 they have made some design choices in the past which now hampers productivity.

> "*We don't think years ahead, but some mistakes were done earlier. It was a bit dumb (...) results in a bit slower productivity."* – PM1

They have recently started using this practice because it has become evident that designing a solution too far ahead is not a good idea. Priorities change, and features that are really important now, might be complete irrelevant in a few months.

Based on the success of using this practice in P2 and the lack of success by not using this practice in P1, my conclusion is that this is indeed an important practice for agile software development and project success.

5.2 Time

Time refers to the amount of time required to complete the project with its scope. The *whole team* practice was the only agile practices related to time which was identified as heavily used.

Whole team

In both projects the teams were assembled to do a certain type of development, and efforts were made to make sure they had the skills to do it. The PM acts as the domain expert and has all the necessary knowledge in order for the developers to create the correct product. Both teams are using the *whole team* practice by ensuring every skill they need is found in the team.

> "*We don't have any designers, but we can get one on request. But the rest of the team should be able to do everything else. That's the principle.*" – PM1
>
> "*We have what we need in the team. When we add new people we make sure they have the competency we require. They should also add something extra to the team.*" – PM2
>
> "*The responsibility goes really high, because you have to find solutions and also make it work. The responsibility and the ability to commit becomes very high.*" – D2

When you empower the team and trust them to get the job done, it motivates them and increases the responsibility of the team. It stands to reason that this will also reduce the time needed to complete features.

The importance of this practice, in terms of project success, is unclear. All respondents had different opinions. The developers found it more important compared to the project managers. Another issue is that this practice might be important in traditional practices as well. Based on this, my conclusion is that this is not one of the most important practices for achieving projects success in agile software projects.

5.3 Quality

Six agile practices designed to increase product quality were identified as heavily used. Table 2 shows quality related practices, which projects they are used in and whether they are used in both projects. These practices are all related to *process quality* which refers to how the system is created and the process around it.

Task boards and *visualize workflow* are heavily used in both projects. *Daily standup, product owner* and *Sprint retrospective* are heavily used in P1. *Improve collaboratively* is heavily used in P2.

Table 2 Heavily used quality related practices

Quality practices	**P1**	**P2**	**Both**
Task board	X	X	X
Visualize workflow	X	X	X
Daily standup	X		
Product owner	X		
Sprint retrospective	X		
Improve collaboratively		X	
Sum 6	**5**	**3**	**2**

Task board / visualize workflow

Task board and *visualize workflow* are very similar practices. The difference is that *visualize workflow* states that progress should be visible, but not necessarily how it should be done. Using a task board is one way to visualize progress. These practices were viewed as the same thing by the respondents. They did not however, agree on how important they are.

They rely heavily on a digital task board in P1. Having a way to see who is working on what, makes the whole iteration more transparent for everybody.

> "*Task board help us visualize the work that we have to do for the particular iteration and where it stands at a particular day or a particular moment.*" – D1

In P2 they have a digital task board *and* a physical white board. The digital board is available to everyone, while the physical board is only available to the developers. The digital board is not updated as frequently as the physical board.

> "*The best thing we are using is the physical white board. Details can be found in the digital board.*" – D2

Most respondents indicated this to be moderately important in order to achieve project success, but they also said it was a really helpful practice to achieve transparency in a team.

> "*It might not be that important for whoever is working on a task (...) more important for others who want an overview of the status of a sprint.*" – PM1

In conclusion this looks like a moderately important practice for achieving project success.

Daily standup

In addition to having a task board, which gives an excellent overview of an iteration, they use the *daily standup* practice in P1. They have daily meetings, normally at the same time each day. They do this to update each other on progress and other relevant information. They consider this an important practice, especially since the PM and the developers are not co-located.

> "*Daily standup is especially important since they (i.e. the developers) are located in Sri-Lanka. Communication is important in all type of work you do (...) regular communication with those you work with (...) to increase communication in the team. Everybody knows what's going on. This is probably reflected in the quality of the product.* " – PM1

They do not have daily meetings like this in P2 anymore. They used to talk every day when they were using Scrum, but now the whole team meets (i.e. video conference) a couple of times a week. The main reason for not doing it daily anymore, is that the team has matured and are more familiar with the domain. There are less questions and impediments. If a developer has a question for the PM he just calls him directly. The PM in P2 emphasizes that they had good experience with daily standup in the past, but he now feels it is an unnecessary time consumer.

The importance of this practice is somewhat unclear. It seems that respondents in P1 finds it very important, while respondents in P2 does not. One likely reason for this is that the PM in P1 also works partly as a developer. This is not the case in P2. This is likely to increase the need for synchronization in P1. It could indicate that this practice is very important when developers are not co-located, but it might not be equally important when all developers sit together and only the PM is located elsewhere.

Product owner

This practice can be viewed in several ways. Having a *product owner (PO)* who manages the backlog is a great way to make sure everybody knows what the priorities are. In addition, this practice is further enhanced if the PO is *dedicated* to this role and/or if there is only one (*single*) PO in the team.

A PO exists on both teams. None of them are dedicated as they have other roles in other teams. In P1 they have a single product owner, while it seems they have more than one in P2.

In P1 they agree that having a *single* PO is important in order to have someone who has the complete overview of current and future requirements, as well as having only one single person responsible for making priorities.

> "*We only have one product owner. I am dedicated in this team, but I also work as a developer in another team. This practice is really important. It creates clarity and makes communication easier. Generally I would say that it helps the whole team since they don't have to make too many decisions (i.e. regarding priorities).*" – PM1

In P2 it was a bit unclear who the PO is. The PM said he was the PO. The developer (D2) however, mentioned two persons when he talked about the PO. It might be just one PO in the team, but additional people with PO authority are also involved.

> *"I have many roles, but I am the PO of the team. It is my responsibility."*– PM2

> *"We have a new PO (i.e. not PM2). He is less available on fixed times, but we communicate more (…) in random time slots."* – D2

The PO in P2 is less available now compared to when they had daily standups and were doing Scrum. There are fewer fixed meetings, so the developers has to contact the PO directly if they have any questions. The developer (D2) said that this is a setup that works. However, it does not sound like they have a single PO. At least there might be some confusion on who has that particular role.

Having a single PO was considered very important for achieving success by both respondents in P1. In P2 they did not consider it important, and this is likely the reason why they do not use this practice to the same extent.

Sprint retrospective / improve collaboratively

Two additional overlapping practices are s*print retrospective* and *improve collaboratively*. The respondents more or less viewed them as the same practice. In P1 they focused on *sprint retrospective*. In P2 they focused on *improve collaboratively*. They did however report the same reasons and benefits from using these practices.

> *"It's important to figure out if productivity is lower than it should be. We use retrospective to figure out these things. It is likely the most important thing we do. "* – PM1

> "*It's (i.e. improve collaboratively) an open discussion where anyone can tell anything regarding what we are doing wrong. What are the pain points? It's very good.*" – D2

It might not be accurate to say that both projects are using the *sprint retrospective* practice *and* the *improve collaboratively* practice, but they are without a doubt trying to improve continuously by discussing things that is not working and trying to come up with ways to improve. They have a slightly different way of doing it, but they all agree that this is an important practice.

If these practices can be considered the same practice, it makes it the only practice where all respondents agree that this is a very important practice in order to achieve project success. Having a way to constantly improve the team and the process is clearly important and stands out as one of the most important agile practices for achieving success.

6 Conclusion

In this chapter I will summarize and conclude findings from the analysis. In addition I will add a section about possible future work and known limitations.

6.1 Conclusion

The goal of this theses was to look for agile practices which contributes to success in agile projects. 53 practices were considered and 23 were found to be heavily used. Out of these, 15 were found to be especially relevant to agile projects and considered important by at least one of the two teams in the study. These are however not considered equally important. Table 3 lists these practices, ordered by how important they are in order to achieve project success.

Five practices were found to be especially important. The most important reason for using *iterative development* and *sprint review* were reported to be early *feedback* from customers. *Iterative development* were also used in order to handle *changing priorities* and *solution uncertainty*, as well as achieving a shorter *time-to-market.*

Incremental design, *stories* and *sprint planning* were used in order to respond to *changing priorities* and *customer needs.*

The principal reason for using *sprint retrospective* and *improve collaboratively* were to improve the process in order to improve the team. The goal of which was to achieve a better working environment and a faster working pace. The consensus was that these are probably the most important practices a team can use.

Ten additional practices were found to be important, but less important than those already mentioned. The reasons for using a *product backlog*, *task board*, *visualize workflow* and dedicated and/or single *product owner* were increased *transparency*, project *overview* and the ability to *plan better*.

Daily standup and *sprint backlog* were mostly used because they improved *communication* within the team. *Team based estimation* and *whole team* were used to increase *estimation accuracy* and improve *development time* respectively.

It is worth mentioning that although some of these practices were concluded to have an importance of two in in Table 3, some might be more important for certain types of agile methods. *Product backlog* and *sprint planning* were reported to be very important by the

Scrum team (P1). This indicated that, which practices are considered most important also depend on which agile method the team is using.

All practices considered especially important are related to *scope* and *quality*. This is not surprising since all respondents are considered *internal stakeholders* and, as described in chapter 3.1, internal stakeholders often use *scope* and *quality* as the most important criteria for determining success.

In conclusion, of the 15 important agile practices, *iterative development*, *sprint review*, *incremental design* and *sprint retrospective / improve collaboratively* are considered most important. These results seem to indicate that practices which improves customer *feedback* and the team *process*, as well as those helping the team to understand *customer needs*, are found very important.

Table 3 Heavily used practices ordered by importance

Practices	**Importance**	**Scope**	**Time**	**Quality**
Iterative development	1	X		
Sprint review	1	X		
Incremental design	1	X		
Sprint retrospective	1			X
Improve collaboratively	1			X
Stories	2	X		
Product backlog	2	X		
Sprint planning	2	X		
Task board	2			X
Visualize workflow	2			X
Daily standup	2			X
Product owner	2			X
Sprint backlog	3	X		
Team based estimation	3	X		
Whole team	3		X	
Sum		**8**	**1**	**6**

6.2 Limitations and future work

This study, like all studies, has some limitations. Firstly, only two cases were investigated. More cases would have provided a more accurate picture of which agile practices are being used, as well as if and why they are important. This would have increased the validity of the study. In addition, only two respondents were interviewed in each case. Additional respondents would have added more detail and insight on how and why a particular practice was being used and how important it was.

Secondly, both cases were very similar. In both projects, the development teams were outsourced to Asia, while the PM was located in Norway. Both teams were very small, with less than five people in each team. The results gathered in this study are therefore only likely to be transferable to teams with similar setups. Practices which is easy to implement in a small team, might be very difficult to implement in a larger team. Outsourcing also has its own set of challenges (Khan, Niazi, & Ahmad, 2009) which is likely to have affected the study.

Because of the limitations set in this study, further research into agile practices by including more cases, more respondents, larger teams and less similarities between the cases, could give new insight. It would also be interesting to see what the results would be if the study was not limited to outsourcing projects.

In addition, further research on the practices identified as most important, and other practices which are likely to affect *customer feedback, customer needs* and *team process* is needed to get a more accurate picture of their importance.

Appendix A – Interview guide

Appendix A contains the interview guide used in the case study.

Part 1 Introduction

In part one, background information on 1) the company, 2) the candidate and 3) the project the candidate has in mind during the interview is collected

Questions: 15-20 minutes

1. Company background information
   a) Please gives some background information on the company.
   b) What type of business is the company? (What is the domain)
   c) Which products do the company have?
   d) Are there any written information available? Product sheets etc.
2. Candidate background information
   a) How long has the candidate worked at this company and/or in this domain?
   b) How many years of agile experience does the candidate have?
   c) How many years of traditional experience does the candidate have?
   d) Which roles has the candidate held in traditional projects? (Developer, project manager, executive, etc.)
   e) Which roles has the candidate held in agile projects?
3. Project background information (Think of one specific project)
   a) What was the name of the product being created? (Project name)
   b) What was the nature of the product? (Why was it created)
   c) When was this project running? Is it still running?
   d) What was the candidate's role in this project?
   e) Which other roles were defined (PO, SM, PM, Developers, QA, UX, etc.)?
   f) How many team members were there in this project?
   g) Who was the target audience?
   h) Was is considered a success?
   i) Anything else to say about the product/project?

Part 2 Practices

In part two, information about the practices which were used in the project is collected.

Questions: 40-60 minutes

4. Open-ended discussion about the practices used in the project.
   - The candidate gives an accurate account of how a cycle/iteration/sprint in the project was conducted. Focus on processes and agile practices.
5. Quantitative survey on agile practices.
   - For each practice in the multiple choice survey, indicate if it was used in this project. (Heavily used, somewhat used, not used, don't know)
6. Qualitative open-ended discussion based on answers given in question 5.
   a) For each of the heavily used practices, why were they used and what was the effect (positive/negative) of using them?
   b) For each of the somewhat used practices, why were they only somewhat used?
   c) For each of the not used practices, why were they not used?
7. Anything else to discuss regarding practices and success factors?

Quantitative Multiple Choice Survey

The multiple choice survey contains a list of practices. The candidate indicates if each practice is *heavily* used, *somewhat* used, *not* used or *don't know.* The list is grouped by the agile methods they originated from [3].

Scrum	
1. Iterative development	8. Daily standup
2. Product backlog	9. Sprint planning
3. Sprint backlog	10. Sprint review
4. Sprint retrospective	11. Product owner
5. Definition of Done (DoD)	12. Scrum master
6. Burn down chart	13. Team member
7. Scrum of Scrums	
XP	
1. Whole team	13. Incremental design
2. Sit together	14. Stories
3. Weekly planning	15. Ten-minute build
4. Quarterly planning	16. Continuous integration

5. Slack	17. Shrinking teams
6. Energized work	18. Root-cause analysis
7. Real customer involvement	19. Shared code
8. Incremental deployment	20. Code and test
9. Team continuity	21. Negotiated scope contract
10. Single codebase	22. Pay per use
11. Daily deployment	23. Informative workspace
12. Test-first programming	24. Pair programming
Kanban	
1. Visualize Workflow	4. Make process policies explicit
2. Limit Work-in-Progress (WiP)	5. Improve collaboratively
3. Manage flow	
Others	
1. Task board	7. Story mapping
2. Coding standards	8. Test-Driven development
3. Refactoring	9. Automated acceptance testing
4. Unit testing	10. Agile games
5. Team-based estimation	11. Single team
6. Integration testing	

Appendix B – Agile Manifesto

Appendix B lists the values and principles defined in the Agile Manifesto.

Agile values

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Agile principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Appendix C – Kanban practices

Appendix C lists the Kanban practices.

1. *Visualize Workflow* - Which is done through a Kanban board which shows each piece of work and how it progresses through the production stages.
2. *Limit Work-in-Progress (WiP)* - Which is managed by putting a maximum count on each column on the Kanban board. If a column contains more tasks than the set upper limit, it is the responsibility of the whole team to fix this so that the flow can continue.
3. *Mange flow* - A Kanban team adds value to an organization in a steady flow. This means that different types of work, which has different workloads, must be tracked in such a way that the flow can be kept over time regardless which work is being done.
4. *Make process policies explicit* - Everybody on the team must have a high level understanding of the entire work process. This is the only way team members can effectively suggest improvements.
5. *Improve collaboratively* - Kanban teams should, thorough measurement and experimentation, reflect and improve on the process continuously.

Appendix D – XP values and practices

Appendix D lists the XP values and practices. The practices are divided into primary and corollary practices.

The 5 XP values:

1. *Simplicity* is a value that focuses on reducing complexity and removing unnecessary features and waste.
2. *Communication* is a value that focuses on making sure every team member knows what is going on and what they are supposed to be doing.
3. *Feedback* is a value where the focus is on getting impressions and feedback early, while there still is time to do something about it.
4. *Courage* is about allowing everybody to work on the same things, and have the courage to show what you have done, so possible improvement can be achieved.
5. *Respect* is the final value and its focus is on recognizing that every team member is important and everybody is accountable for successes and failures

The 13 XP primary practices:

1. *Whole team* – The team members have all the skills and connections it needs to succeed in a project.
2. *Sit together* – All members of a team sits within eye contact of each other.
3. *Pair programming* – Two programmers work together to solve a problem.
4. *Informative workspace* – Important information about the project is put on the walls for easy access and direct exposure.
5. *Weekly planning* - Make plans for visible and valuable progress every week
6. *Quarterly planning* – Create quarterly themes that should be addressed during the weekly planning.
7. *Slack* – Add optional items when planning, in case of delays.
8. *Test-first programming* – Code by writing a failing test. Then write code to make the test pass.
9. *Incremental design* – Don’t plan to long ahead. Design the system for *today’s* requirements.
10. *Stories* – Plan business functionality as incremental user stories. A short and simple description which is written from the user perspective.
11. *Ten-minute build* – Create automatic build systems that can test as much of the system as possible in ten minutes.
12. *Continuous integration* – Code changes should be automatically deployed to a testing site as often as possible.
13. *Energized work* – Keep a live outside the office, so you bring energy to your work.

The 11 XP corollary practices:

1. *Real customer involvement* – Get feedback from the customer every week. This is to ensure the product is being created the correct way.
2. *Incremental deployment* – Don’t deploy the whole system at once. Deploy in steps. Smaller deployments are less error prone.
3. *Team continuity* – Teams that work well together should stay together.
4. *Shrinking teams* – When teams get more experience and becomes more effective, they can produce more. Reduce the size of the team rather than increasing the workload.

5. *Root-cause analysis* – Fix mistakes by learning from them and prevent them from happening again. Don't make quick fixes, but find the root cause of the problem.
6. *Shared code* – All parts of the code can be changed by anyone in the team. No code "belongs" to a specific developer.
7. *Code and test* – Code and tests are the main documentation of the product and should be treated as such.
8. *Single codebase* – All team members should work on the same code base. Avoid long lived code branches. Merging causes conflicts.
9. *Daily deployment* – Deploy code one a day. Shorter cycles between each deployment, results in faster customer feedback.
10. *Negotiated scope contract* – Time, cost and quality can be fixed. Negotiate on scope and agree on shorter contracts with smaller releases. This makes it easier to respond to changing requirements.
11. *Pay Per Use* – Charge the customer every time the product is used instead of every time a new release is done.

Appendix E – Scrum practices

Appendix E shows a list of commonly accepted scrum practices. The Scrum Alliance has not defined any specific list of practices.

1. *Burn down chart* – Chart that shows the implementation progress in a sprint.
2. *Daily standup* - Daily meeting where the team meets to share information and discuss potential obstacles. Normally time boxed to maximum 15 minutes.
3. *Definition of Done (DoD)* – An agreed upon set of criteria that has to be true in order for a user story to be completed and considered done.
4. *Iterative development* – Development is broken up into small time boxed iterations called sprint. Each sprint usually lasts 2 to 4 weeks.
5. *Product backlog* - The most important action item resides on the top of a prioritized feature list.
6. *Product owner* (PO) – One person who is responsible for the product backlog and decides which tasks has priority.
7. *Scrum Master (SM)* – Servant leader. Responsible for making sure the process is followed and that everything works well in the project.

8. *Scrum of Scrums* – For large groups, the teams are divided into smaller agile team. Each team conduct daily standup internally. In addition one person from each team participate in a daily standup where all teams are represented.
9. *Sprint backlog* – Action items selected for development in the current sprint.
10. *Sprint planning* – Before each sprint is started, the team and the product owner meets to decide which items from the *product backlog* should go into the *sprint backlog*.
11. *Sprint retrospective* - At the end of each sprint, the team looks back and determine if anything about the process can be improved.
12. *Sprint review* - After each sprint is complete, the changes are shown to stakeholders who can give feedback.
13. *Team members – Team member* is the only other defined role apart from PO and SM.

Appendix F - Agile practices

Appendix F gives a short description of the agile practices which where identified in chapter 3.4. There are some overlap between practices in appendix F and appendix C, D and E. The description is gathered from the Agile Alliance guide to agile practices [1].

1. *Agile games* – Techniques for estimating the complexity of a user story.
2. *Automated acceptance testing* - Acceptance test is a formal description of the expected result of implementing an action item. The test is run automatically regularly and alerts the developer if a test fails.
3. *Behavior-Driven Development (BDD)* – An augmented version of test-driven development (TDD). More focus on communication, business outcomes and collaboration.
4. *Coding standards* - Rules and conventions on how code should be written in order to easy understand it and increase quality.
5. *Collective code ownership* - Every team member is allowed to change any part of the code, regardless who initially wrote it.
6. *Continuous deployment* - Similar to continuous integration, but each action item is put directly in production upon completion.

7. *Continuous integration* - Every time an action item is completed, the test site is automatically updated and the new change can be tested.
8. *Daily standup* - Daily meeting where the team meets to share information and discuss potential obstacles. Normally time boxed to maximum 15 minutes.
9. *Dedicated product owner* - Each development team has a single product owner who decides which action items is most important.
10. *Integration testing* – A phase in testing where groups of software modules are tested together to see if they interact properly.
11. *Iteration planning* - When a new iteration starts, the team makes a plan and decides which action items to work on for the next iteration.
12. *Iteration reviews* - After each iteration is complete, the changes are shown to stakeholders who can give feedback.
13. *Kanban* - Task board where each column only can hold a limited number of action items.
14. *Open work area* - Members of a team sits together in order to facilitate easy information sharing and collaboration.
15. *Pair programming* - Two developers sit together and implements a solution for a single action item.
16. *Prioritized backlogs* - The most important action item resides on the top of a to do list.
17. *Refactoring* - The process of re-writing existing code in order to improve it.
18. *Release planning* - Long term plan to roughly plan which action item should be developed in which iteration.
19. *Retrospective* - At the end of each iteration, the team looks back and determine if anything about the process can be improved.
20. *Short iterations* - Development is done in cycles (iterations). A short iteration is normally 1 to 4 weeks.
21. *Single team* - Development and testing is done by the same team.
22. *Story mapping* - Action items are ordered along two independent dimensions, priority and implementation sophistication.
23. *Task board* - Board with status columns, where each action item in a sprint is placed based on its status.

24. *Team-based estimation* - All team members discuss and agree before an action item is estimated.
25. *Test-Driven development (TDD)* - Create unit tests for an action item before the item is developed. The test will fail until the action item is fully implemented. Forces a developer to think before implementing code.
26. *Unit testing* - Short programs that executes a narrow part of the production code and check if it work as intended.

References

Agarwal, N., & Rathod, U. (2006). Defining 'success' for software projects: An exploratory revelation. *International Journal of Project Management, 24*(4), 358-370. doi:http://dx.doi.org/10.1016/j.ijproman.2005.11.009

Agile Alliance. Agile Manifesto.   Retrieved from http://agilemanifesto.com

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management, 17*(6), 337-342. doi:http://dx.doi.org/10.1016/S0263-7863(98)00069-6

Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading, Mass: Addison-Wesley.

Beck, K., & Andres, C. (2004). Extreme Programming Explained: Embrace Change. *Reading: Addison-Wesley Professional.*

Blank, S. (2013). Why the Lean Start-Up Changes Everything. *Harvard Business Review, 91.5*(May), 63-72. Retrieved from https://hbr.org/2013/05/why-the-lean-start-up-changes-everything

Boynton, A. C., & Zmud, R. W. (1984). An assessment of critical success factors. *Sloan Management Review (pre-1986), 25*(4), 17-27.

Bryde, D. J., & Robinson, L. (2005). Client versus contractor perspectives on project success criteria. *International Journal of Project Management, 23*(8), 622-629. doi:http://dx.doi.org/10.1016/j.ijproman.2005.05.003

Chappell, D. (2013). *The Three Aspects of Software Quality: Functional, Structural, and Process*. Retrieved from

Charette, R. (2005). Why software fails. *IEEE Spectrum, 42,* 42-49.

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *The Journal of Systems and Software, 81*(6), 961-971. doi:10.1016/j.jss.2007.08.020

Cohn, M. (2003, 2015-12-22). Introducing An Agile Process to an Organization. Retrieved from http://www.mountaingoatsoftware.com/articles/introducing-an-agile-process-to-an-organization

Cohn, M. (2010a). The Role of Leaders on a Self-Organizing Team. Retrieved from http://www.mountaingoatsoftware.com/blog/the-role-of-leaders-on-a-self-organizing-team

Cohn, M. (2010b). *Succeeding with agile: software development using Scrum*. Upper Saddle River, N.J.: Addison-Wesley.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9–10), 833-859. doi:http://dx.doi.org/10.1016/j.infsof.2008.01.006

Griffiths, M. (2012). *PMI-ACP Exam Prep*: RMC Publications, Inc.

Hammarberg, M., & Sundén, J. (2014). *Kanban in Action*: Manning Publications Co.

Jacobsen, D. I. (2005). *Hvordan gjennomføre undersøkelser? : innføring i samfunnsvitenskapelig metode* (2. utg. ed.). Kristiansand: Høyskoleforl.

Jalali, S., & Wohlin, C. (2012). Global software engineering and agile practices: a systematic review. *Journal of Software: Evolution and Process, 24*(6), 643-659. doi:10.1002/smr.561

Khan, S. U., Niazi, M., & Ahmad, R. (2009, 13-16 July 2009). *Critical Success Factors for Offshore Software Development Outsourcing Vendors: A Systematic Literature Review.* Paper presented at the Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., . . . Zelkowitz, M. (2002). Empirical Findings in Agile Methods. In D. Wells & L. Williams (Eds.), *Extreme Programming and Agile Methods — XP/Agile Universe 2002* (Vol. 2418, pp. 197-207): Springer Berlin Heidelberg.

Mar, K., & Schwaber, K. (2002, March 22, 2002). Scrum with XP. *informit.com*.

Mens, T. (2008). *Introduction and roadmap: History and challenges of software evolution*: Springer.

Mistra, S. C., Kumar, V., & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. *The Journal of Systems and Software, 82*(11), 1869-1890. doi:10.1016/j.jss.2009.05.052

Poppendieck, M., & Poppendieck, T. D. (2013). *The Lean Mindset*: Addison Wesley.

Project Management Institute. (2013). *A guide to the project management body of knowledge: (PMBOK guide)*. Newtown Square, Pa.: Project Management Institute.

Project Management Institute. (2014). PMI Today January 2014. *PMI Today*.

Project Management Institute. (2015). PMI Today January 2015. *PMI Today*.

Serrador, P., & Pinto, J. K. (2015). Does Agile work? — A quantitative analysis of agile project success. *International Journal of Project Management, 33*(5), 1040-1051. doi:http://dx.doi.org/10.1016/j.ijproman.2015.01.006

The standish Group. (2013). *CHAOS Manifesto*. Retrieved from

VersionOne. (2015). *9th Annual State of Agile survey*. Retrieved from

Wicks, A. M., & Roethlein, C. J. (2009). A Satisfaction-Based Definition of Quality. *The Journal of Business and Economic Studies, 15*(1), 82-97,110-111.

Williams, L., Brown, G., Meltzer, A., & Nagappan, N. (2011, 22-23 Sept. 2011). *Scrum + Engineering Practices: Experiences of Three Microsoft Teams.* Paper presented at the Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.

Wysocki, R. K. (2014). *Effective project management: traditional, agile, extreme*. Indianapolis, Indiana: Wiley.

Yin, R. K. (1994). *Case study research: design and methods* (2nd ed.). Thousand Oaks, Calif.: Sage.

Yin, R. K. (2014). *Case study research: design and methods* (5th ed.). Thousand Oaks, Calif.: Sage.

Web pages:

[1] Agile Alliance, guide to agile practices. http://guide.agilealliance.org/. Accessed 2015/09/17.

[2] Version One, State of Agile Survey. http://stateofagile.versionone.com. Accessed 2015/11/4.

[3] Quantitative survey: https://goo.gl/0TJYtY. Accesses 2015/11/1.

[4] Scrum Alliance, Scrum Guide. https://www.scrumalliance.org/why-scrum/scrum-guide. Accessed 2015/12/28.