NTNU
Norwegian University of
Science and Technology

Whole Team Thinking and Success Factors in Large Scale Agile Development

An Exploratory Case Study

Tina Christin Syversen

Master of Science in Computer Science
Submission date: May 2016
Supervisor: Torgeir Dingsøyr, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Agile software development has been the preferred method since it was introduced as a software development method in 2001. It was originally intended for small co-located teams but in later years it has experienced widespread acknowledgement also in large scale development projects. One thing that is not known is how to best achieve success in these kinds of projects.

In this thesis we have explored two different cases of large scale projects. Because of the complexity which arise in large scale projects it is difficult to know how they can be conducted in the best possible way. The number of people involved, the need for coordination, and locating the different teams are some of what complicates a large scale agile development project and it is therefore hard to know how to best achieve success when considering these new complexities. From the literature research we therefore found several possible success factors, and the factor whole team thinking became the focus. The reason for this is that there is a difference when working in a project with one team and a project with several teams which depend on each other. Whole team thinking is about all teams thinking in the same direction and having the same goals throughout the projects, e.g. having shared mental models across all teams involved. These factors were investigated in the two cases and it was discovered how they affected the different cases; Omega and Tellus.

From the results we could see that whole team thinking was one of the factors holding the two projects together. This was noticable when the teams in the project considered the fact that they were working together and not competing against each other. When the teams noticed how important it was to collaborate they were more helpful toward the other teams and they understood the whole project more than when they just thought about their own team and their own part of the project. When they did not consider the project they were more quick to blame the other teams and hinder the project's progress.

Keywords: Agile, large scale, multiteam system, shared mental models, success, whole team thinking.

Sammendrag

Smidig utvikling har vært den foretrukne metoden siden den ble introdusert som en metode i programvareutvikling i 2001. Opprinnelig så var den tenkt for små samlokaliserte team, men i de senere år har det blitt mer og mer utbredt også i større prosjekter med flere team. Noe som har vært uvisst er å finne ut hvordan man kan oppnå like stor suksess i de store prosjektene som i de små.

I denne studien har vi sett på to ulike stor-skala prosjekter. På grunn av kompleksiteten som oppstår i stor-skala prosjekter er det vanskelig å vite hvordan disse kan lykkes på best mulig måte. Antall personer involvert, behovet for koordinering og det å lokalisere alle teamene er noe av det som kompliserer stor-skala smidige prosjekter og som gjør det vanskelig å vite hvordan en kan oppnå suksess når en vurderer disse aspektene. Fra litteraturen har vi derfor funnet flere mulig suksessfaktorer, og vi har valgt å fokusere på ”whole team thinking”. Denne faktoren går ut på at alle team skal tenke i samme retning og ha de samme målene gjennom hele prosjektet, for eksempel ved å ha felles mentale modeller på tvers av alle teamene. Disse faktorene har blitt sett sammen med de to casene; Omega og Tellus, og hvordan de påvirket prosjektene.

Fra resultatene kan vi se at ”whole team thinking” var en av faktorene som holdt prosjektene samlet. Dette var spesielt merkbart når teamene ikke konkurrerte mot hverandre, men jobbet sammen. Når teamene så hvor viktig det var å samarbeide var de mer hjelpsomme med tanke på andre team og de forsto helheten i prosjektet og tenkte ikke bare på sin egen del av prosjektet. Når de ikke tenkte på helheten så var de raskere til å skylde på hverandre når ting ikke fungerte og hindret fremgangen til prosjektet.

Nøkkelord: Smidig, stor-skala, multiteam systemer, felles mentale modeller, suksess, helhetlig tenkning

Preface

This paper was written as the master thesis in my final year on the Master Program in Computer Science at the Norwegian University of Science and Technology, NTNU. My specialization has been on software systems, where I have learned much from the different courses taken over the years. Agile development has been interesting through these years and I have been able to use this knowledge in some of the project courses, and in my summer job for Avanade in 2015.

The reason for choosing the subject of success factors in large scale agile was because I felt there was possible to contribute something. This area is still relatively new and therefore it is exciting to research something not many else have had the possibility of discovering yet.

This author would like to thank the supervisor of this paper, Torgeir Dingsøyr. With his help I was able to investigate actual cases, and without his support and knowledge on this subject this thesis would not be possible. There is also necessary to thank some other students who have also contributed. The thesis of Espen Andreassen was helpful throughout this project because of the knowledge of the subject and Omega case, and the observations of Aravinthan Yogarajah was helpful in the investigation of the Tellus case. The work with this thesis is related to the Agile 2.0 project funded by the Research council of Norway through grant 236759, and supported by the companies Kantega, Kongsberg Defence & Aerospace, Sopra Steria, and Sticos.

Trondheim, May 31, 2016

Tina Christin Syversen

Table of Contents

List of Tables

List of Figures

Abbreviations

NTNU	=	Norwegian university of science and technology
IT	=	information technology
QA	=	quality assurance
UI	=	user interface
UX	=	user experience
XP	=	extreme programming
IS	=	information systems
PO	=	product owner
CPO	=	chief product owner
APO	=	area product owner
PPO	=	proxy product owner
EPO	=	epic product owner
FPO	=	feature product owner
SM	=	scrum master
TDD	=	test driven development
COP	=	communities of practice
MTS	=	multiteam system
LeSS	=	large scale agile (framework)
SAFe	=	scaled agile framework
DAD	=	disciplined agile delivery

Chapter 1

Introduction

The first chapter is a short introduction of the motivation behind this thesis together with the problem description and background. The scope with its limitations, contribution and the target audience will also be presented. In the end there will be a brief report outline which explains what to expect in the forthcoming chapters.

1.1 Motivation

Agile software development methodologies have been widely accepted since the introduction of the agile manifesto in 2001 [1]. This transformation witnessed the introduction of several software methods, tools and techniques. Agile methods were said to:

> *"best suit co-located teams of about 50 people or fewer who have easy access to user and business experts and are developing projects that are not life critical"* [2]

Even so companies have started applying agile practices to large scale projects.

When applying agile in large scale projects it is necessary to sort out requirements with potential dependencies across the different teams. This creates new forms of coordination mechanisms that do not exist in small scale [3]. At XP2013 and XP2014 Dingsøyr and Moe conducted a workshop about the principles of large scale agile development which resulted in a research agenda in 2013, and a revised research agenda in 2014. One of the topics with the highest priority included variability factors in scaling. This involved identifying what factors are important in large projects that influence the development process [2].

Today there are several projects which are working agile in large scale. One example of this is Spotify [4]. At Spotify they have implemented a matrix structure and they change their way of working constantly to find the best solutions for their business. SAP is another example of a company working in large scale [5]. What we do not know exactly is how these companies can work agile in large scale and achieve success with their methodology.

The challenges which arise in large IT-projects are many. The number of people, dependencies and locations are just some of what needs to be considered. With so many new varying conditions it is difficult to discover how and why they can inhibit a project's success. The teams should also communicate and coordinate with each other, which can be difficult because agile teams are supposed to be self-organizing [6]. This indicates that there is a difference when applying agile to large projects. This is an area where there is little information for those applying agile to large projects.

This master thesis has therefore examined the different factors improving the project success in large scale agile development. The focus has been on the factors and in which way these assist the project to success. Because of this we have studied some of the large organisations and projects which have implemented agile techniques, together with two cases that the researcher have been lucky enough to follow during the duration of this thesis.

The reason for choosing this subject is because little is known about it today. It is exciting exploring a field where much is still undiscovered and to hopefully contribute something to the cause. This thesis might not change much today, but it might bring a new outlook on large scale agile in the future.

1.2 Problem Description and Background

There are today several different organizations and projects that benefits from agile practices, some of these are Spotify [4] and SAP AG [5]. Because of the benefits agile methodologies have proven to offer SAP AG, the world's leading producer of enterprise software have transitioned from a waterfall-like process model to an agile development model [5].

The revised research agenda for large scale agile software development by Dingsøyr and Moe [2] had high priority on the topic of factors in scaling. After studying several lessons learned-articles it was clear that there were many different success factors that were applicable in both large and small scale agile. Misra et al. [7] and, Chow and Cao [8] have researched the potential success factors in small scale agile. These have therefore been compared with both the lessons learned-articles, and the cases involved in this thesis.

There is not much information on this topic in existing papers, which is probably why it was a high priority topic on the revised research agenda by Dingsøyr and Moe [2]. This is why the topic is highlighted in this thesis which focus on different success factors in large scale agile development projects.

A project is usually a planned set of interrelated tasks to be executed over a fixed period and within certain costs and other limitations[1], but in this thesis the project can also last

[1]http://www.businessdictionary.com/definition/project.html

for an unknown amount of time, e.g.: Spotify which is always improving to better fit the customers requirements. The purpose of this master thesis is:

- To add to the body of knowledge on large scale agile project
- To solve a problem by improving a large scale agile project
- To find out what happens when regarding the factors in a large scale agile project
- To find evidence to inform practice of how to best conduct a large scale agile project
- To develop a greater understanding of people and their world in a large scale agile project
- To contribute to other people's well-being in a large scale agile project

The end product will therefore be a combination of these purposes [9].

This thesis will be an ”in-depth study of a particular situation” [9] where we will focus on two different cases. In addition it will be ”an exploration of a topic, area, or field” with focus on the different factors in large scale agile, together with a more thorough investigation into large scale agile projects.

1.3 Scope and Limitations

This thesis is about possible success factors and whole team thinking as a success factor in large scale agile development projects. The time limit on this master thesis has been twenty weeks. All of the work for this thesis has been completed during those weeks, in addition to some of the material being found and researched during the preliminary paper [10], which also focused on success factors and whole team thinking. The preliminary paper consisted in finding the research question and material for that, and is where the success factors were found. The preliminary paper were therefore more of a research study while this thesis only uses the information gathered there together with new information, and it is compared with the two cases; Omega and Tellus.

Because of the time constraints it was not possible to get more material from the two cases. With more time it might have been possible to conduct interviews in one or both of the cases, which could have shed more light on the research question and success factors. With more time it would have been possible to delve deeper into the cases, and maybe even follow other relevant cases. It could have been possible to ask more about how they are affected by the different factors, if they think about them at all. This was difficult because of the necessity for an ongoing or newly concluded project, preferably close by.

Since this research field, large scale agile, was fairly new, there was not that much information on the subject, which is another reason of why it was chosen. The field had not been investigated thoroughly according to Dingsøyr and Moe [2] in a conference paper from XP2014, where variability factors in large scale agile development was proposed as a future research field. This thesis therefore gives an overview over what has been found in other papers, both about agile development, large scale agile, and the different factors found in the different papers. This thesis is not about how to best scale an agile development project, but it is about the factors that can help large scale projects heavily influenced by the whole team thinking factor which will be explained later in this thesis.

The problems with large scale agile projects is the fact that an agile team is supposed to be self-organized [11]. In a project with several self-organized teams this would complicate the project, which implies that they can only be self-organized to a certain degree. The number of people involved increases the chances for dependencies and it is difficult for everyone involved in one project to not communicate and coordinate with each other. Spotify has over 30 teams across three cities [4]. The fact that they are too many to be placed in one location suggests the complexity of working in a large scale agile project.

The focus on this thesis is therefore on the possible success factors. These are found in other journals and lessons learned-articles, and compared with the two cases. We will discuss these factors individually and together with the factor whole team thinking which is the main focus. We will see how they affect the two cases both positively and negatively, and how they can help the projects to achieve success. We can therefore see the difference in how necessary they are, and how they affect the different projects. The scope for this thesis is therefore about what factors that need to be considered to achieve success in a large scale agile project, and how they can be used for the best results.

1.4 Contribution

The contribution this thesis will provide is to the topic of success factors in large scale agile. After suggesting possible success factors we will try to display them in the two case studies and how they have affected these projects. We will compare the two different cases and see how each of them have solved the factors in their project. We will see which factors are applicable to both projects, and therefore factors which might be applicable to most projects and which factors depend on the case. In the end we will focus on the factor whole team thinking and see how this affects both the cases and the other factors found.

1.5 Target Audience

The target audience for this thesis is primarily people interested in large scale agile projects. It is for those conducting large scale agile projects that are interested to know how they can best achieve success. This is relevant especially for the two cases followed and also for the supervisor and the Agile 2.0 project. This paper will be available through NTNU which indicates that other students can use it for inspiration either they write about agile or large scale agile development. For those who practice agile in large scale projects this thesis can help in understanding how to best conduct these types of projects by reading about the success factors and how they are implemented in the cases. For scientists in large scale agile how the success factors influence the projects in both positive and negative is something that could be interesting regarding other research fields in the future.

1.6 Report Outline

Chapter 1 Introduction:
This chapter is to give a brief overview of the thesis, this includes the motivation, background, scope, contribution and target audience.
Chapter 2 Theory:
The theory chapter brings a listing of agile software development and what lies in whole team thinking and shared mental models. Large scale is given an explanation and there is an overview over the different success factors found in different articles.
Chapter 3 Method:
Here the literarture review is carried out, with the selection strategy and research question. In addition this chapter contains how the data collection and analysis were perfomed and how thesis was completed.
Chapter 4 Results:
The results chapter holds the findings from the two cases. These are seen in context with the success factors from the theory chapter.
Chapter 5 Discussion:
The results are discussed further and the two cases are compared. The focus here will be on whole team thinking alone and together with the other factors. An evaluation of the study is also conducted.
Chapter 6 Conclusion:
This chapter includes the conclusion of this thesis in conjunction with the research question. This chapter also collects the loose threads.
Chapter 7 Future Work:
In this chapter suggestions for future work in large scale agile development is given.

Chapter 2

Theory

The theory chapter will present the relevant literature for this thesis. First we will briefly explain agile software development together with one of its methods. After this we will give an introduction to large scale agile before presenting the different success factors found, together with an explanation. In the end we will delve deeper into one of the factors and shared mental models.

2.1 Agile Software Development

In this section we will introduce agile software development and discuss how it is different from the waterfall-model. We will introduce one of the methodologies scrum, before considering team perfomance in an agile project. In the end we will focus on earlier research on agile software development.

2.1.1 Agile Methods

In the late 1990's there were several methodologies that had started gaining increased public attention, each with a different combination of new and old ideas. What these had in common was the close collaboration of the development team and the business stakeholders[1]. When seventeen software development practitioners gathered in 2001 they tried to find a common ground[2]. From this the Agile Software Development Manifesto[3] emerged, which was based on four values:

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

[1]https://www.agilealliance.org/agile101/what-is-agile/
[2]http://agilemanifesto.org/history.html
[3]http://agilemanifesto.org/

While there is value in the items on the right, the left ones are valued more.

Agile software development offers a professional approach to software development that encompasses human, organizational, and technical aspects of software development processes [12]. Agile software development is an umbrella term for a set of methods and practices based on the values and principles from the agile manifesto. The word ”agile” by itself indicates that something is flexible and responsive, so agile methods implies its ”ability to survive in an atmosphere of constant change and emerge with success” [8]. Agile methods are people-centric, recognizing the value of competent people and what their relationships bring to software development [13]. It focuses on providing high customer satisfaction and extensive documentation are of little value to the developers. The key point is that agile approaches plan features, not tasks, as the first priority because that is what the customers understand. Agile approaches recommend short iterations in the two-to-six weeks range. By having dynamic prioritization the customer can always reprioritize their features at the end of an iteration [14]. At the core of agile practices is the idea of self-organizing teams where the members work at a pace that sustain their creativity and productivity [1]. This implies that the team is responsible for the whole product or service, that they control the members' task behaviour and have the authority to make decisions about task assignment and work methods [15].

Agile methods are, according to Cohen, Lindvall and Costa [9], a reaction to the traditional ways of developing software, and acknowledge the ”need for an alternative to documentation driven, heavyweight software development processes”. It is the focus and values behind the agile methods that differentiate them from the traditional methods. The waterfall model, see figure 2.1[4], was a way of assessing and building for the users' needs. The model starts with a complete analysis of user requirements, then engineers establishes a definitive and exhaustive set of features etc. which are all well-documented. Next phase is the design phase before the programmers implement the well-documented design and the system is tested and shipped. Problems with this model was the fact that users changed their minds and that requirements changed. The waterfall model therefore worked best when change rates were low. Other incremental and iterative techniques took the process behind waterfall and repeated it through the development lifecycle [9].

Agility, ultimately, is about creating and responding to change. It is not the practices that they use that are new, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability [14]. Agile approaches is often used in the development of web-based applications [16]. As an iterative and incremental method to software development, the agile method is implemented in a very interactive manner to make good quality software that answers the alternating needs of its users.

According to Scott Ambler [6] there are five criterias that determine whether a team is agile or not. It is important to note that these are just suggestions and that different projects might require different methods. For example; some agile teams produces working soft-

[4]http://www.sharonencyclo.com/waterfall-model-sharon-encyclopaedia/

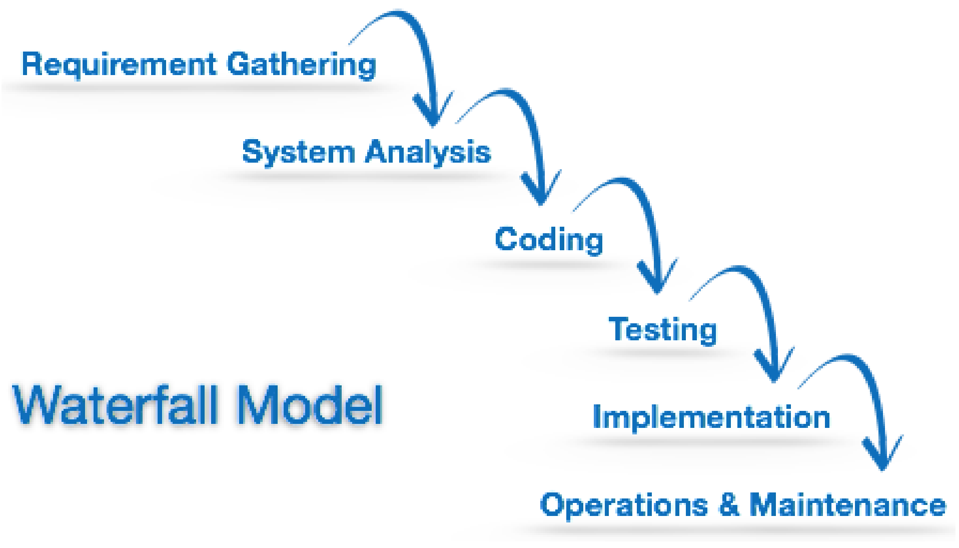

Figure 2.1: The Waterfall Model

ware every two weeks, while others might not do this as often because of a more complex situation. The first criteria is that the agile team provides a value to their stakeholders at a regular basis. The second is the validation criteria. This implies that for agile teams it is necessary to do continous developer regression testing and that the disciplined agile teams take a test driven development approach to validate their work. Active stakeholder participation is the third one. Here the importance of working closely with the stakeholders, ideally on a daily basis, is recommended. The fourth one is self-organization and that is the criteria that is the most challenging to fulfill. Agile teams are self-organizing, and disciplined agile teams work within an appropriate governance framework. The last and fifth criterion is improvement. The agile teams have to regularly reflect on, and disciplined teams should measure, how they work together, and then act to improve on their findings in a timely manner.

Agile software development has followed the general trend in product development organizations towards a team-based structure with the idea of self-organizing teams [17]. Teamwork has been found to be a key driver to successful software development and it provides a better solution to organizational problems. Agile team behaviour might affect the ”ability to make the necessary modifications in order to meet new challenges” [17]. It is said that members of effective teams are aware of team functioning and of changes in the team environment. This team-based structure can be seen in several of the agile methodologies. According to a world-wide survey [18] with professional software developers completed by VersionOne in 2012 the two most widely used agile methods among professional software developers today are scrum and extreme programming (XP).

Scrum

In 1996 scrum was defined as a process that "accepts that the development process is unpredictable," and formalizing the "do what it takes" mentality [19]. A scrum is a team of eight individuals in rugby. Everyone in the pack acts together with everyone else to move the ball down the field. The entire team have one single focus, work closely together and has clear priorities [20]. The focus of scrum in software development is project leadership and requirements management [6].

Scrum projects are divided into iterations, also called sprints, which lasts for two-to-four weeks, see figure 2.2[5] for an overview of the scrum cycle. Each sprint consists of a sprint planning session, daily stand-up meetings, sprint demo and a sprint retrospective. In the sprint planning the product backlog, a list of prioritized items from the product owner, is discussed and items are taken from this and into the sprint backlog[6].

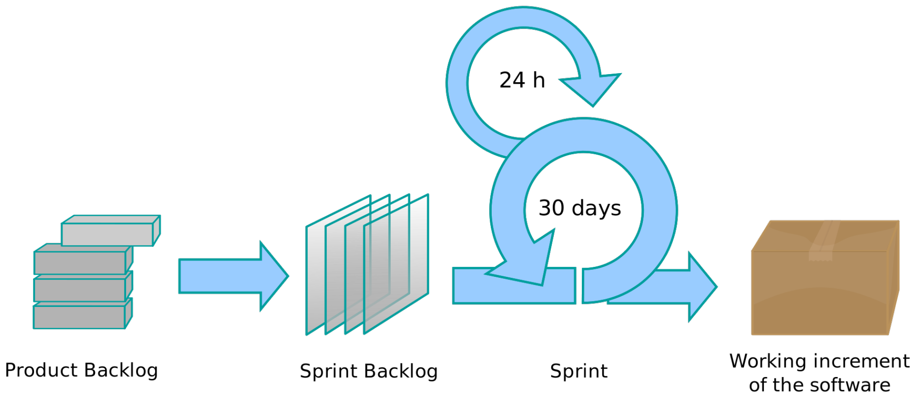

Figure 2.2: The Scrum Cycle

The daily stand-up meetings are held everyday, involves the whole team and is fascilitated by the scrum master. There is usually three questions that is answered during these meetings [20]:

- What have you done since last time?
- Are there any obstacles in your way?
- What will you do next?

These meetings usually last about 15-30 minutes and are not used to brainstorm ideas or solving the problems they have [20]. In the end of a sprint a demo will be held. Here the

[5] http://www.rapidsoftsystems.com/agile-development-process.html
[6] https://www.scrumalliance.org/why-scrum

Table 2.1: The Different Scrum Processes

Process	**What**
Product backlog	An overview of the requirements requested by the customer/ PO which is prioritized after each sprint
Sprint	An iteration which lasts between two-to-four weeks where there hopefully will be produced working software
Sprint backlog	Requirements at the top of the product bakclog which are to be completed during the sprint
Sprint planning	Planning of the sprint where requirements are taken from the backlog, divided into tasks and is given a duration
Daily stand-up	A meeting every day at approximately the same time where everyone explains what they are doing
Demo	A presentation at the end of the sprint of the product to the customer/ PO
Retrospective	A discussion of how the sprint was conducted with positive and negative items which focus on how to conduct the next sprint better

different stakeholders usually meet and they are shown what the team have managed to produce during the sprint. After this there might be reprioritizing of the product backlog. In the end the team has a retrospective where they discuss how the sprint went, and how the next sprint can be conducted even better than the last. For an overview of the different artifacts in scrum see table 2.1.

Table 2.2: The Different Scrum Roles

Scrum roles	**Who**	**What**
Product owner	A person from the customer with an interest and knowledge in the product	Manages customer requirements and communicates these to the team
Scrum master	One team member with leader responsibilities	Facilitator between PO and team, and a team member
Team	Software engineers, architects, programmers, analysts, QA experts, testers, UI designers	Responsible for developing the product

A scrum team consists mainly of three different roles. These are the: product owner, scrum master and team[7]. The product owner is responsible for managing customer requirements and communicating these to the team. The scrum master is the fascilitator between the product owner and the team, and works to remove any impediments that are obstructing the team from achieving its sprint goals. The team is a small development team

[7]http://scrummethodology.com/

responsible for self-organizing to complete work, with a mix of software engineers, architects, programmers, analysts, QA experts, testers and UI designers. The team is normally not more than ten people. For an overview of the different roles see table 2.2.

2.1.2 Team Performance

A common definition of a team is:

> *"a small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves mutually accountable"* [21]

Collaborative work in teams can promise the potential of greater adaptability, productivity and creativeness as compared to individual work [17]. Teamwork is at the core of agile development [22]. In agile development it is necessary for a team to be adaptable because of the everchanging needs of the customer, and it has been found that a team's adaptability is a key determinant of team effectiveness [23].

Teams have the potential to provide complex, innovative, and comprehensive solutions to organizational problems [24]. There are several models available but it is important to distinguish between team performance and team effectiveness. Team performance accounts for the outcomes of the team's actions regardless of how the team may have accomplished the task. Team effectiveness takes a more holistic perspective in considering both how the team completed their task and how the team interacted to achieve the team outcome. The important factors of teamwork found by Salas et al. [24] is the ones called the "Big Five", see figure 2.3. These core components of teamwork include team leadership, mutual performance monitoring, backup behaviour, adaptability and team orientation. These components have three coordinating mechanisms which are: shared mental models, closed loop communication and mutual trust.

In 2013 Dingsøyr and Lindsjørn [21] conducted eighteen focus group sessions to investigate what factors agile software practitioners perceive to influence effective teamwork. They chose to use the research-based Salas et al. model of the "Big five" as a basis for this research. The questions asked during the focus groups was: "What fosters effective teamwork?" and "What hinders effective teamwork?". In total there were 1183 items that were placed in the eight teamwork component groups, see table 2.3. As we can see from the table shared mental models was one of the components that received the most items on fostering team performance, together with closed-loop communication and mutual trust, all of which are the coordinating mechanisms of the model.

The components were clustered into three main groups after they were reviewed:

1. Team leadership and closed loop communication
2. Shared mental models, team orientation and mutual trust
3. Mutual performance monitoring, backup behaviour and adaptability

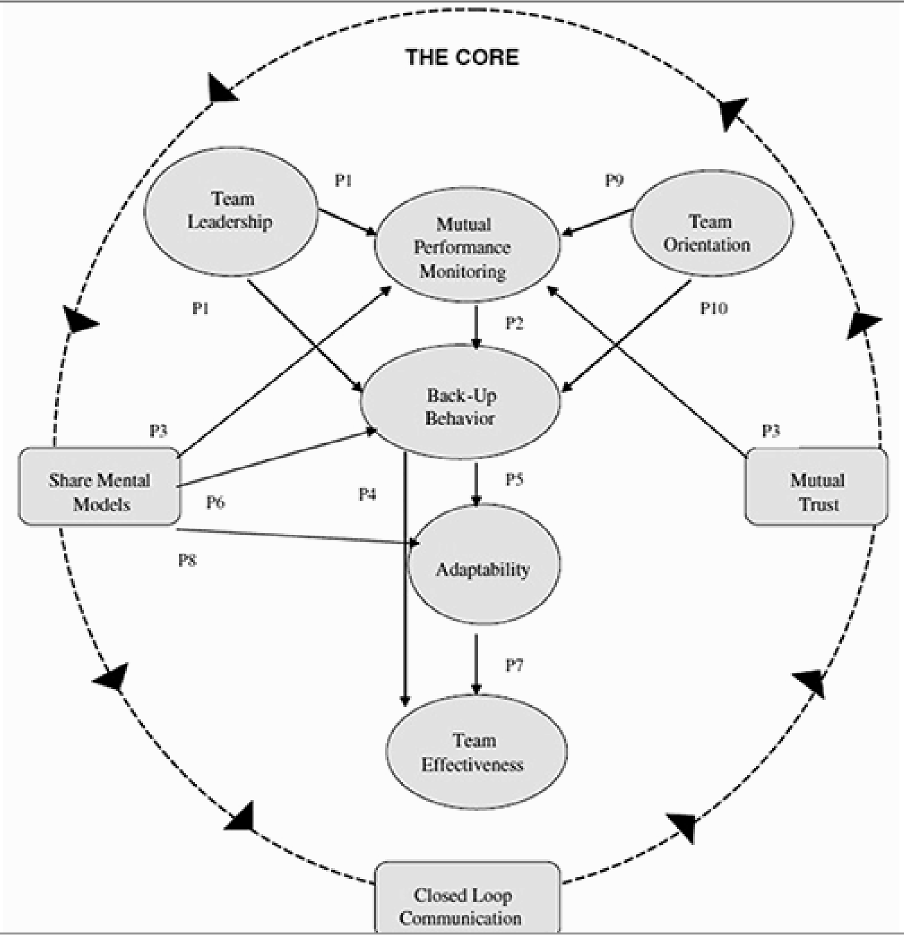

Figure 2.3: Salas Big Five

Table 2.3: Summary of Distribution Between the Teamwork Components

Team Component	**Foster**	**Hinder**	**Total**
Team leadership	90	139	229
Mutual performance monitoring	49	22	71
Backup behaviour	44	57	101
Adaptability	46	50	96
Team orientation	91	65	156
Shared mental models	104	59	163
Closed-loop communication	97	58	155
Mutual trust	122	90	212
Sum	**643**	**540**	**1183**

The research confirms the fact that teamwork is of high relevance to agile software development teams. Agile teams are said to be self-managing, while the ”Big Five” model is not particularly tailored for self-managing teams. Even though we can see that the model is applicable for the self-organizing teams in agile software development.

If we focus on shared mental models team leadership is influenced by this in how the team leader has a role in the creation, maintenance and accuracy of the team’s shared mental model. For mutual performance monitoring it is important with shared mental models because it provides teammates an understanding of what other team members are supposed to be doing. Shared mental models are necessary antecendents to effective backup behaviour because they form the foundation for decisions of when a team member must step in to provide backup, who should step in, and what assistance is necessary.

As we can see from figure 2.3 there are ten research propositions, where three of these come from shared mental models. These are:

> **P3:** Effective mutual performance monitoring will only occur in terms with adequate shared mental models and a climate of trust.
>
> **P6:** Effective backup behaviour requires the existence of adequate shared mental models and mutual performance monitoring.
>
> **P8:** Effective availability requires the existence of adequate shared mental models and effective engagement in mutual performance monitoring and backup behaviour.

These propositions between the variables represents the interrelationships of the ”Big Five”, and as we can see the ”Big Five” cannot function without their coordinating mechanisms, which include shared mental models.

2.1.3 Research on Agile

In 2008 Dybå and Dingsøyr [25] published a systematic review of empirical studies on agile software development. The objective of the study was:

1. What is currently known about the benefits and limitations of agile software development?
2. What is the strength in support of these findings?
3. What are the implications of these studies for the software industry and research community?

They identified 36 empirical studies on agile software development where thirty-three were primary studies, and three secondary studies. Most of the studies reviewed that investigated agile projects dealt with employees that were new to agile development and they were also published in conferences. According to Dybå and Dingsøyr [25] the research published before 2005 revealed a lack of theoretical and methodological rigor.

Some of the benefits reported by the studies from Dybå and Dingsøyr [25] was customer collaboration, work processes for handling defects, learning in pair programming, thinking ahead of management, focusing on current work for engineers and estimation. One of the recurring themes found was human and social factors and how these factors affect agile development methods. One problem with the studies was that the strength of evidence were low, which made it difficult to offer specific advice to the industry.

In 2012 Dingsøyr et al. [1] did a literature search on the ISI Web of Science where they identified 1551 research papers on agile software development published from 2001 to 2010. The majority of the articles originated in the US, Canada. and Western Europe, but there were 63 countries in total on all continents which had had agile development as a research theme. Most of the articles published came from conferences, most notably the International Conference on Agile Software Development.

Nerur and Balijepally [13] revealed in 2007 that the progression of ideas in agile software development were remarkably similar to conceptual pattern shifts in software design. They looked at agile software teams as holographic organizations. Everything which is in the whole hologram can be accessed through any of its parts since each one is a reflection of the whole. Agile philosophy facilitates their formation of holistic teams through a culture that encourages the changing of roles and jobs based on independence.The team's skills enable it to function even when some members are unable to perform [13]. This can be explained by the fact that the agile teams are self-organized and therefore are less dependent on one single person.

2.2 Large Scale Projects

In this section we will introduce the concept of large scale projects before delving deeper into Spotify, which is an example of a large scale agile project. After this we will discuss multiple teams working together and how to locate the teams in a large scale project. In the end we will discuss different frameworks and different ways to scale a large agile project.

2.2.1 Large Scale Agile

The success and wide use of agile software development has inspired use in new domains and companies have been increasingly applying agile to large scale projects [26], see figure 2.4 [27]. The discussion on the ability of agile practices to scale to larger software development efforts has been widely debated in recent years [28]. It is still important to remember that agile is not a tool-based technique that can easily be rolled out across large organizations, it is value based and needs buy-in [29].

When describing agile development in everything from large teams to large multiteam projects that make use of principles of agile development in a whole organization, the term ”large scale agile development” has been used [26]. The definition given by Dingsøyr et al. [26]is the following: ”agile development efforts with more than two teams”. In addition they define very large scale as ”agile development efforts with more than ten teams”.

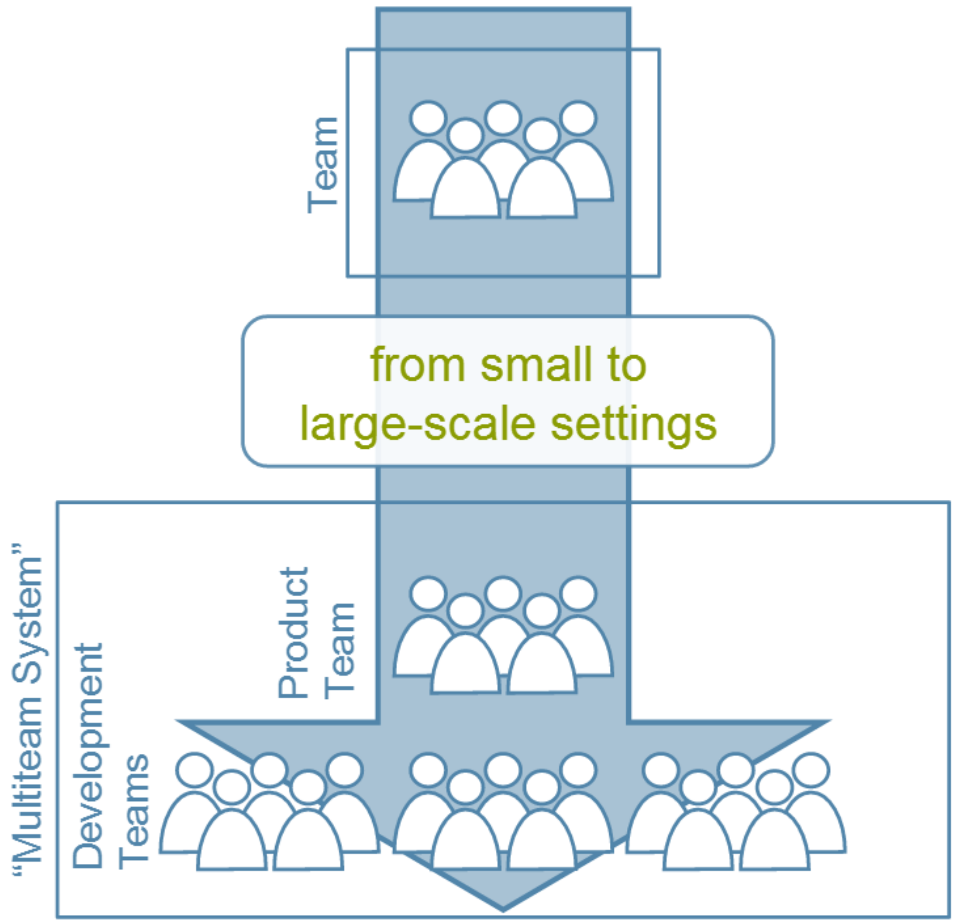

Figure 2.4: From Small to Large Scale

With these definitions we exclude agile methods applied in large organizations, also called ”enterprise agile”. For more information about this see appendix C.1.2.

When large teams are supposed to produce software functionality quickly, the agile methods must scale to meet the task [30]. There has been said to be eight scaling factors for agile [16; 6]. These are: team size, geographical distribution, regulatory compliance, organizational complexity, technical complexity, organizational distribution, domain complexity and enterprise discipline. These scaling factors are ranges and all will not be applicable to any given project [16; 6]. According to Saeeda et al. [16] there are two terms when discussing scalability of agile. These are ”scaling out” and ”scaling up”. Scaling up is concerned with the use of agile methods for developing large software systems that cannot be developed by smaller teams. Scaling out is more concerned with how agile methods can be introduced across a large size project with many years of software experience [16].

When deciding how to divide the teams in large scale agile it is common to divide into feature teams or component teams [31]. When using feature teams the teams are grouped into areas that are customer- and not architecture-centric. Component teams divide their work based upon the system architecture. When choosing which way to divide the teams the customer should be in focus. The reason for this is the fact that when choosing component teams the requirements and backlogs are more technical of nature, this indicates that the product owner should be technically savvy or the team will have to translate technical requirements into needs for the product owner [31]. The mix of feature and component teams is one of the reasons for occuring inter-team dependencies [32].

2.2.2 Example: Spotify

Spotify is a good example of scaling agile, see figure 2.5 [4]. In an article from 2012 the scaling at Spotify is explained by Kniberg and Ivarsson [4]. It is important to note that Spotify is a company which is evolving fast and this article by Kniberg and Ivarsson is just a snapshot of how Spotify worked at the time of the article, and there have most likely been several changes since then.

At Spotify a squad is the same as a scrum team and it is designed to feel like a mini-startup. The squad has a long-term mission which can be e.g.: building and improving the android client. Because they are on a long-term mission they become experts in that area. The squads have a workspace which includes a desk area, lounge area and a personal ”huddle” room. Most of the walls are whiteboards. Each squad is encouraged to spend about ten percent of their time on ”hack days” where they can do whatever they want. They have a product owner which is responsible for prioritizing the work conducted by the team, and the product owners of different squads collaborate with each other.

A tribe is a collection of squads that work in related areas, like the music player. The tribes have a tribe lead and they are physically in the same office. They are based on the concept of the ”Dunbar number” which implies that people cannot maintain a social relationship with more than about one hundred people. This is because of the increase in restrictive rules, bureaucreacy and other things that could get in the way. The tribes hold

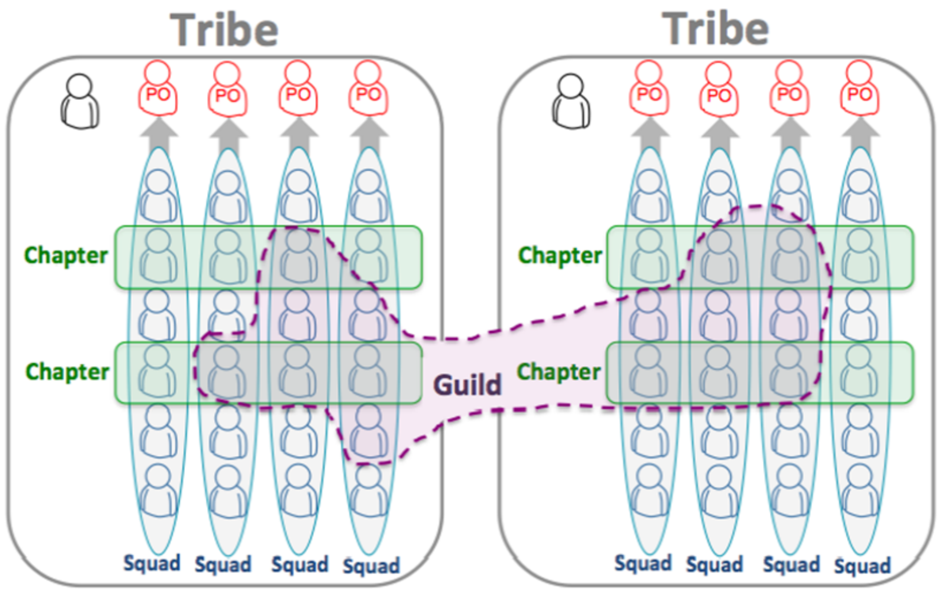

Figure 2.5: An Overview of Scaling at Spotify

regular gatherings which is an informal get-together.

Dependencies is helped by the squads regularly being asked which squads they depend on and if these dependencies are blocking or slowing them down. Scrum of scrums happen only on demand because the squads are fairly independent. Chapters and guilds are used to glue the company together. A chapter is a family of people which have similar skills and are working in the same general competency area within the same tribe. They meet regularly but are not necessarily evenly distributed across the squads. The guild is referred to as more of a ”community of interest” where those who want can share knowledge, tools, code and practices. Guilds cut across the whole organization.

Anyone is allowed to edit the system, but to make sure that someone focuses on the integrity of the whole system they have the role ”system owner”. All systems have one or two system owners, when there are two; one has a developer perspective and the other an operations perspective. This person is not responsible for doing the job, but for making sure it is implemented. Normally this person is a squad member or chapter lead with other day-to-day responsibilities.

2.2.3 Multiple Teams

When there are multiple teams working together it is often called a multiteam system. A multiteam system (MTS) is:

”two or more teams that interface directly and interdependently in response to

> environmental contingencies toward the accomplishment of collective goals" [33; 34; 35].

This is when there are several independent teams that work together in a team of teams, or multiteam, setup [34; 35]. Typically this structure has been based on the modularised component of the software architecture. In such a setting each team owns the development and maintenance responsibility for one component [35]. The collective goal of this system can be broken down into a goal hierarchy and constitutes a key characteristic of any multiteam system [34]. All teams within a multiteam system share at least a distal goal[8], while the individual teams pursue their more proximal goals[9] [34; 36]. This goal structure leads to teams displaying input, process and outcome interdependence with at least one other team [35; 36].

MTS have received growing attention in organizational psychology over the last decade, but the aspect of coordination is underdeveloped acoording to Scheerer et al. in different articles [33; 34]. This indicates that only the different areas and linkages have been explored. Cross-team processes have later been proved to have the most value in MTSs with a high independent goal hierarchy. This implies that well-managed MTS processes was influenced positively, but it did not support the team-level action processes.

Coordination is the management of dependencies. Inter-team coordination is the coordination of activities between two or more teams within an MTS [15]. There are two coordination types. The first is explicit coordination which consists of mechanistic elements like plans and rules. The other is implicit coordination which consists of cognitive elements like shared mental models and team expertise [37]. In a study of leadership in MTSs the leader teams were trained in two ways by either facilitating strategy development or coordination [33]. The strategy training was positively related to explicit coordination while coordination training affected the implicit coordination much stronger. This indicates that an unidentified mechanism like shared mental models seemed to influence inter-team coordination [33].

2.2.4 Co-Located vs Distributed

One of the scaling factors presented by Ambler [11] is geographical distribution. When working in a large scale project it is difficult to co-locate everyone in the same place, and some projects even go across several countries. Vlietland and van Vliet [38] present three main issues arising with distribution of software practices:

- Spatial separation
- Time zone differences
- Cultural differences

[8]A long-term goal, in this case the end product
[9]A short-term goal, here the sprint goals

Spatial separation leads to exacerbation of communication and coordination. Time zone differences can make it difficult for teams to meet, and it can therefore make it harder to receive answers to questions and dependencies. Schnitter and Mackert [39] propose that time zone differences of more than three hours makes it difficult to agree on common time for daily scrum meeting, and this implies scheduling meetings in general. Cultural differences is about the culture in the teams. Teams in different countries might have another way of working than others do. In the lessons learned-article from Scnitter and Mackert [39] they mentioned that multicultural teams typically broke up within two months mainly because the role of the scrum master is interpreted very differently among European, Asian, and American people. A scrum master from one culture found it hard to meet the expectations of team members from another culture.

Larman and Vodde [40] recommends co-located teams, but open up for the possibility that the teams can be in different sites. It is more important to avoid a single dispersed team with scattered members. Martini, Pareto and Bosch [41] found ten root factors and recommendations for them. One of the root factors was ”No co-location”. With large scale it is almost impossible not to distribute teams because of the lack of space, and even the distance of one floor can make the teams distributed. They therefore recommended that especially teams that interact more intensely should be located closer.

2.2.5 Frameworks and Scaling Methods

In a response to the rapid adoption of agile software development in large scale settings, several frameworks have been propsed by practitioners and agile evangelists. The most known are: Large Scale Scrum (LeSS), the Scaled Agile Framework (SAFe), the Nexus Framework and Disciplined Agile Delivery (DAD). Out of these frameworks Nexus is the closest to scrum with a lot of the same practices and processes on an inter-team level with a group of scrum teams, called a nexus. Nexus is said to be the exoskeleton of scaled scrum[10].

LeSS, see figure 2.6[11], provides two different large scale frameworks; LeSS and LeSS Huge which can have up to a few thousand poeple on the project. Most of the scaling elements are focused on directing the attention on all teams over to one product instead of ”my part”[12]. LeSS introduces the concept of requirement areas as a means to group teams by areas of strongly related customer requirements headed by area product owners. All teams are in a common sprint to deliver a shippable product every sprint. Some of what distinguishes LeSS from normal scrum is the fact that there are two sprint plannings; one for everyone and one for each team, and the overall retrospective. The overall retrospectives purpose is to explore improving the overall system rather than focusing on one team.

[10]https://www.scrum.org/Resources/The-Nexus-Guide
[11]http://agileatlas.org/articles/item/large-scale-scrum-more-with-less
[12]http://less.works/less/framework/index.html

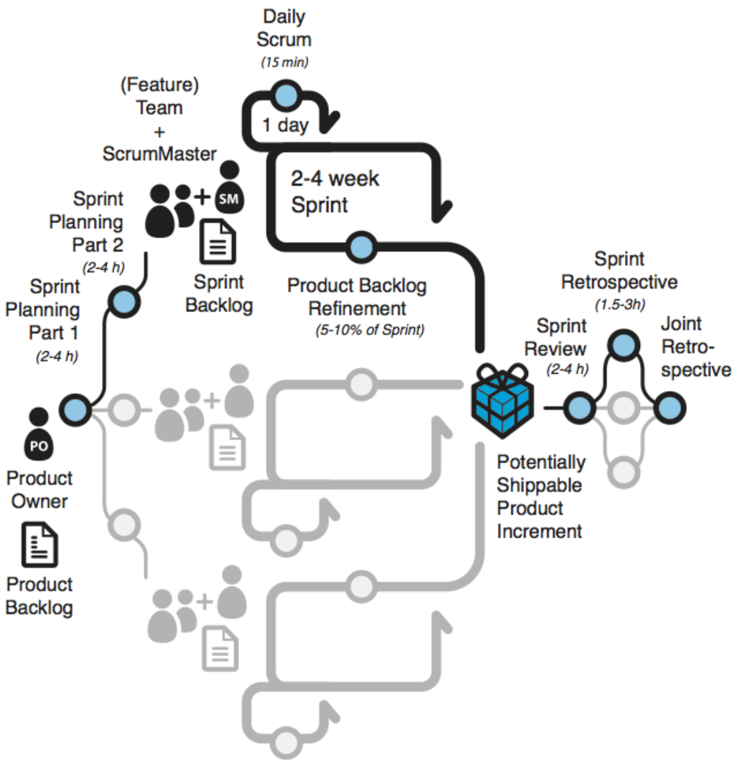

Figure 2.6: LeSS Framework up to Ten Teams

SAFe is based on a number of immutable, underlying lean and agile principles[13]. It upholds four core values: alignment, built-in-quality, transparency and program execution[14]. SAFe is a scaled agile application in the enterprise and targets seven areas to achieve parallell scrum development [16]. These seven targets are:

- Cross-functional teams
- Standardized planning and tracking
- Standardized iterations
- Smaller, frequent releases

[13]http://www.scaledagileframework.com/safe-lean-agile-principles/
[14]http://www.scaledagileframework.com/safe-core-values/

- Concurrent testing
- Continous integration
- Regular reflection and adaption

SAFe is based on a number of newer paradigms in modern system and software engineering. This includes lean and system thinking, product development flow and agile development[15].

DAD is a hybrid framework that builds upon the solid foundation of other methods and software process framework[16]. The approach combines scrum with the best practices from multiple methodologies like XP, Rational Unified Process, Kanban, etc. The focus of DAD is on the delivery and a full product lifecycle goes from the initial idea, through delivery, to operations and support. See figure 2.7[17] for an overview of the scaling factors in DAD.

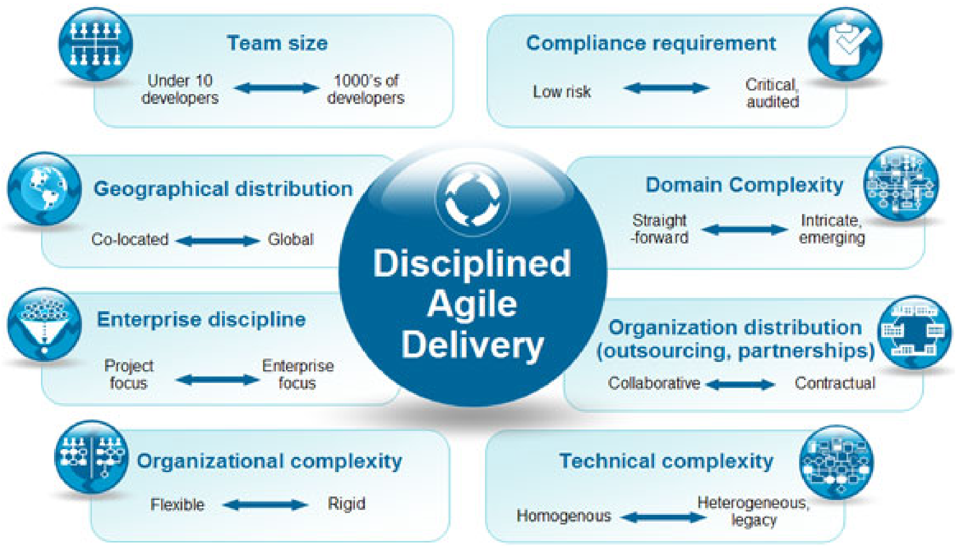

Figure 2.7: DAD Scaling Factors

The frameworks concentrate on the practices, but they do not say much about the delivery mode, planning and coordination. In November 2015 Scheerer and Bick had a presentation in Trondheim about ”Five Ways to Scaling Agile Coordination in Large-Scale Agile Software Development” [27]. Scheerer and Bick both work in SAP and have written several articles based on the scaling there [23; 33; 34; 35; 36]. In the presentation they

[15]http://www.scaledagileframework.com/lean-agile-mindset/
[16]http://www.disciplinedagiledelivery.com/introduction-to-dad/
[17]http://www.ibm.com/developerworks/rational/library/automate-software-development-processes/

Table 2.4: The Five Scaling Cases by Scheerer and Bick

Case	**Delivery Mode**	**Planning**	**Coordination**
Traditional	4-week sprints 1-year release cycle	Central teams assign epics to development teams	Teams via PO
Cloud	2-week sprints 1-month release cycle	Virtual CPO/PO round	Teams via POs, SMs and developers
Distributed	2-week sprints 4-month release cycle	Central architecture team proposes technical solutions for requirements	Teams via POs and central architecture team
Co-located	4-week sprints 3-month release cycle	Quarterly release planning workshop with all members	Teams via POs or developers
Modular	4-week sprints 3-month release cycle	Individual POs plan work packages in accordance with CPO	Teams occasionally via POs

explained five different scaling methods, see table 2.4, and how they can be coordinated.

The first case is the traditional case with scaling through central team directives. Here there are about thirteen teams with one hundred and forty people at four locations. They are component-based, which indicates that they are business process-oriented. In this case there is a central team which assigns epics via top-down directives but there are little bottom-up feedback from the development teams. Cloud is the second case with ten teams and about one hundred and five people at two locations. The scaling here is via iterative proxy collaboration. The organizational structure is feature-based and the solution to problems are often merely reported to CPOs (chief product owners).

The third case is distributed which involves seven teams with about seventy people over six locations. This is scaling via central planning based on team inputs. This case is component-based and software module-oriented. They have a central architecture team and the planning is based on feedback from team architects and POs. Co-located is the fourth case and is scaling through full collaboration. Here there are six teams with approximately 85 people in one location. This case is another one which is component-based and business process-oriented. They have regular information-sessions between selected teams and maintenance of a detailed wiki. The fifth and last case is the modular case which is scaling via ad-hoc communication. There are four teams with around forty people over three locations and they are feature-based. Here there is a high involvement of, and guidance by, the CPO.

As we can see there are several ways of scaling a large scale agile projects. In addition to finding a framework it is necessary to find out how to coordinate the teams: bottom-up or top-down. Number of people and locations therefore have to be considered together with the influence from the management. When scaling agile methods it is important to scale them without sacrificing the underlying principles of the agile manifesto [30].

2.3 Success Factors

In this section we will introduce the different possible success factors for this thesis, but first we will introduce the background for these factors.

2.3.1 Introduction

In 2014 Dingsøyr and Moe held a workshop during XP2014 [2]. Here they found a revised research agenda for large scale software development. From here the original research question for this thesis appeared:

"What are the possible success factors in large scale agile development?"

This could appear to be a broad topic since there might be a number of factors that can make a project successful. After researching and reading through several papers, see appendix A, there were a lot of possible ideas as to what could be a success factor. A factor was categorized as a success factor if it was:

- Important for the project's success
- Mentioned as a missing piece
- Mentioned as a recommendation
- Mentioned as a problem or challenge

The factors found can be seen in table 2.5.

All factors come from at least one article, most come from either a conference article or journal article, while some are lessons learned. The factors in lessons learned might only have been applicable in that case, but since they are mentioned they are important in some degree. Most of these factors are cited in multiple articles which gives support to both conference and lessons learned-articles. Some of the factors are only applicable in large scale while most are applicable in both large scale and small scale. The way they affect the project, and in what degree, can be somewhat different for some of the factors. The factors with the least data according to the research question, those marked with * in the table, can be found in appendix D.

Table 2.5: Success Factors in Small and Large Scale Agile

Success factor	**Small scale**	**Large scale**
Customer collaboration	X	X
Agile software engineering techniques and agile evangelist	X	X
Responding to change	X	X
Team	X	X
Leadership	X	X
Planning*	X	X
Testing*	X	X
Governance*	X	X
Divide after you conquer*		X
Continous integration*		X
Bridgehead		X
Co-location		X
Product owner		X
Whole team thinking		X

2.3.2 Customer Collaboration

The customer is an important part of agile methodologies since the customer is involved the whole way opposed to just the beginning and the end. In Chow and Cao's [8] survey study they had a list of twelve possible critical success factors. One of these were customer involvement and in the end of the article they state the importance of strong customer involvement. In agile it is important for the customer to understand agile principles, if not the road can be a bit bumpy as experienced in the article from Koski and Mikkonen [42]. The effort that the customers have to put in an agile project is much more essential than in other projects, and several customers do not understand this before undertaking an agile project. The product owner is a good example of customer collaboration. The product owner is one of the most critical roles in scrum, and difficult to implement successfully [31]. The product owner is therefore the main contributor, but can in addition be the main bottleneck because of this [42]. When beginning an agile project it is therefore necessary to explain to the customer what is expected from them.

Another important thing when it comes to customers is involving the right people, or the key stakeholders [43]. It is essential to identify the correct stakeholders, if not the whole project can fail. It is not necessarily the leaders that are the correct stakeholders, but the people who are going to benefit from the system. For understanding the system goals and features it is important to know which stakeholders that are target for the system's features and functions. The customer should be able to prioritize scenarios or requirements if they are to be satisfied with the product, if not they might not get all the necessary requirements they need to do their job. Factors like customer satisfaction, customer collaboration and customer commitment is significant to the project success [16]. This applies to both small and large scale, and we will describe the importance of scaling the PO-role more in

2.3.9.

2.3.3 Agile Software Engineering Techniques and Agile Evangelist

One of the three critical success factors for agile software development projects presented by Chow and Cao [8] is agile software engineering techniques. The team has to practice rigorous agile software engineering techniques. Not all projects that are supposed to practice agile have practiced it before, and it is therefore necessary to have top-notch agile consulting [44]. This can be accomplished by an agile evangelist who coordinates training and rollout. Early coaching is important and especially when no one have practiced agile earlier.

Benefield [45] explains how they introduced agile development to Yahoo. There the pilot teams received some early coaching from leading agile thinkers like Ken Schwaber, Paul Hodgetts, Mike Cohn and Esther Derby. As the program expanded they established an internal coaching team at Yahoo to evangelize the benefits of scrum throughout the company, and to train coaches and support the teams that wished to use it. Since they did not have enough capacity at a time they experienced that teams that went ahead without the training failed and performed mini-waterfalls instead of being agile. The coaches, or agile evangelists, should be people with strong skills in collaboration and building consensus. Yahoo is an example of enterprise agile and therefore it is stated that finding good people who really understand the agile principles, and training them to help their own teams, are keys to scaling effectively in a large organisation. Even though scaled agile is highlighted it is still important for any agile team to be familiar with agile techniques and practices.

2.3.4 Responding to Change

In essence, agility means responding to changes quickly and efficiently [46]. Daneva et al. [47] accentuates the necessity to embrace change, especially when (re)prioritizing requirements. This necessity have an impact on how agile requirements prioritization happens in both large and small projects together with project constraints. Responding to change is one of the four values which constitutes the essence of agile development methods [36]. Agile development beautifully handles changes in regard of the client's market advantage [16]. Being respondent to change is one of the things that makes agile so customer friendly, and displays how it can adopt to changing business situations [46]. There are two types of changes different events can lead to. First-order changes are caused by events leading to incremental adoption, while second-order changes lead to brief periods of upheaval, which change the systems deep structure [34].

2.3.5 The Team

Chow and Cao [8] identified three critical success factors in agile software projects, where one of them were team capability. In agile development it is the self-organizing scenario-based team which is important for the project's success [11; 48]. The reason for the self-organizing teams being a success might have something to do about the fact that the best people for planning are the ones who are going to implement it [11]. Scrum gives teams more power to make decisions concerning development speed and quality, which have lead to wide acceptance among teams [39]. Self-organization is therefore one of five criterias that Ambler [6] proposes to determine whether a team is agile or not, and in addition it is where most teams fail.

For successfully adopting the principals and practices of agile software development the team members should posess the ability to think and act like ”Global Citizens” or ”Tribe Members” [49]. This is especially important in large scale where the primary challenge is finding the right people to form this ”tribe”. This is recommended by Dr. Dobb, one of the pseudonyms of Scott Ambler, that the easiest way to scale agile techniques is to hire good people [50]. As mentioned by Moore and Spens [49] their number one rule is to not base your decision on agile experience when finding people to a large scale agile project. Even though people are good at agile, it does not imply they are good when there are suddenly multiple teams opposed to just one team.

Another important aspect is that it is important to avoid the ”super team” [43]. The super team is not necessarily healthy to the success of a project. This is because they tend to create thick silos within which they do great work, but can totally ignore anything outside this silo. Instead of focusing on delivering business value to their customer, they focus on the success of their part of the system. Other problems with the super team is the fact that they might have to much work because there is no other team with their abilities and they can end up slowing down the whole process. The scenario-based teams help mitigate the formation of a super team since in order to make a scenario work, team members collaborate on interfaces across multiple components and technologies [43].

Optimizing the collaborative teamwork is more important than optimizing individual productivity in agile development [51]. Schmidt et al. [18] found, in a case study at SAP, that there is a higher motivation in the team and a better common understanding among the team members when they do pair programming. Moore and Spens [49] mention that especially in large scale projects it is necessary for team members to participate in cross-team activities and many teams then struggle with balancing their identity as a team member against their membership in the larger project.

According to Scmidt et al. [17] in their paper about team adaptability they propose that agile team behaviour might affect the ”ability to make the necessary modifications in order to meet new challenges”. There are differences in how the team is a factor in accomplishing success in small and large scale agile development, but it is still one of the most important aspects in both. Even though there are several teams in large scale they are still self-organizing, and it is still important to hire good people. The difference in people might be the fact that in small scale it is an advantage to know agile practices, while in large scale this could be a disadvantage because they forget to look outside their own team.

2.3.6 Leadership

A leader is the person who drives the project. The leader in an agile development team is often the scrum master, if scrum is the chosen methodology. A product owner can also be seen as a sort of leader. This is because the product owner is the person who has the most influence on the project since he or she usually comes from the customer and has a customer-view. We will discuss the product owner more in 2.3.9. According to Erik Moore [52] the key to successfully scaling agile was finding leaders who understood as Grace Murray Hopper said:

”You manage things; you lead people”[18]

The organization structure may influence certain leadership behaviours, but not all.

The overall project manager is responsible for planning the whole project and making sure that eventual sub-projects are going in parallell in a timely manner [53]. In small scale it is usually only called the project manager, while in large scale it might be necessary to have one overall project manager and several sub-project managers. This is depending on the size of the project where there could be multiple levels in the hierarchy. According to Moore [52] it was important that leaders exhibited the following behaviours:

- Leading instead of managing
- Flexing team boundaries
- Driving both team and project success
- Balancing team and individual needs

This implies that it is necessary for a leader to set direction for their teams, recognize the necessity to maximize team velocity, understand that one team’s success does not drive the project, and recognize that a team consists of individuals.

Because of the increase in size there are some new factors that appear in large scale agile projects. This could be project management teams which have the overview of the entire project. These are necessary to hold the project together and to make sure that the teams are aware of steps taken by the other teams.

2.3.7 Bridgehead

Having a bridgehead between the teams is extremely important in large scale because it helps with the coordination among them. Coordination is long recognized as one of the fundamental problems of software engineering [54]. This is one of the root factors proposed by Martini et al. [41] where they recommend that a bridgehead can help coordination and that face-to-face communication is benefical. This is especially important when locating dependencies among the teams. As Moore and Spens [49] mention it is important in large scale that team members participate in cross-team activities and allocate time to project-wide activities. They explain that the team technical leads are required to dedicate significant portions of their time outside the team room. This can be compared with Spotify [4] where the system owner takes a ”system owner day” to do housekeeping work on

[18]She was a computer programmer that helped develop a compiler that was a precursor to the widely used COBOL language - http://www.biography.com/people/grace-hopper-21406809

the system, see 2.2.2.

One of the different coordinating mechanisms that has been proposed are the scrum of scrums, also called metascrum, which have been presented by several papers [5; 26; 34; 36; 38; 55]. Whether the scrum of scrums actually work have been researched by Paasivara et al. [56]. Their findings concluded with the fact that scrum of scrums worked poorly. Contrary to these findings Dingsøyr et al. [26] found scrum of scrums to be a well-functioning coordination mechanism in the Omega case. In scrum of scrums the scrum masters of the different teams brief the other teams what they are working on. It has the same function as the stand-up meeting, but instead it is about the teams and not individuals in a team. In some cases people from other teams attend the stand-up to ensure information exchange between teams everyday [53].

Another coordination mechanism is the communities of practice (COP). A community of practice is a ”group of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on ongoing basis” [57]. They have three important characteristics which are domain, community and practice. Communities of practice can provide a long range of both long-term and short-term benefits to both the organization and to the members of the community. At Spotify they have ”chapters” and ”guilds”, these are explained in 2.2.2. They meet regularly to discuss their area of expertise and their specific challenges. A guild resembles more a community of practice where people who want can share knowledge, tools, code and practices which cuts across the whole organization.

Dingsøyr et al. [26] saw in their research of the Omega case that coordination with a large number of arenas made coordination efficient. Some of these arenas were:

- Board discussions
- Experience forum
- Instant messaging
- Lunch seminars
- Rotation of team members
- Technical corner

The rotation of team members is also recommended by Elshamy and Elssamadisy [53] as a practice to help with problems of consistency and duplication. This can be compared with dependencies. There are normally dependencies between teams that work on the same project and a bridgehead, or cross-team activities, is therefore an important aspect in large scale to discover and fix these dependencies.

2.3.8 Co-Location

It is said that agile methods ”best suit co-located teams of about 50 people or fewer who have easy access to user and business experts and are developing projects that are not life-critical" [26]. In large scale there might be multiple people and it becomes increasingly difficult to place everyone at the same location. Geographical distribution is presented as one of the scaling factors by Ambler [11]. Even being in different cubicles within the same building can erect barriers to communication, let alone being in different cities or even on different continents. The distance can lead to delays and lack of communication and commitment [41]. In Schnitter and Mackert's papers on SAP [5; 39] they mention co-locating all product team members as one of the prerequisites for a product team to function properly. In addition they considered it essential to bring all team members together in one location during project kick-off meetings. Even though it is necessary with co-located teams it is more important to avoid a single dispersed team with scattered members [40]. One of the recommendations by Martini et al. [41] is that teams that have to interact more intensely should be located closer.

When Giblin et al. [29] interviewed the development teams at the beginning of the agile introduction the open plan area provoked negative responses. After working with agile methods for a period of time, the necessity for co-location was strongly recognized by the developers. A good example of locating teams that need to work more intensely together is Spotify [4]. They have scaled to over thirty teams across three cities. As mentioned in section 2.2.2 the squads working in related areas are all physically in the same office, most often next to each other. The lounge areas they have are there to promote collaboration between squads.

2.3.9 Product Owner

The product owner (PO) is the person who prioritizes the requirements and provides detailed information about the business to developers [6]. In scrum it is the product owner that makes the decisions [58]. The product owner is the key contributor, but can also be the main bottleneck [42]. The PO owns all the requirements and communicates all the functional and non-functional specifications to the developers. The PO role is pivotal in providing a bridge between the development and project management team. It is therefore necessary to have the right person in this task [48]. This is the same for both small scale and large scale. But the difference when it comes to large scale is the necessity of scaling the PO role. One of the recommendations from Koski and Mikkonen [42] was to split the responsibilities of the PO for a number of persons. This was to be able to communicate with both the customer and teams effectively.

A single PO is not able to work with all the teams [31]. The number of teams one PO can be responsible for varies in the different papers. Paasivara et al. [31] have found that Pichler recommends that a PO works with no more than two teams simultaneously, while Larman and Vodde [40] thinks a single PO can have up to ten teams. Several articles suggest the use of area product owners (APO) [5; 31; 39; 40]. APO concentrate on specific areas of the product. The areas can be compared to the tribes at Spotify [4], see 2.2.2. There is normally one overall PO and several APOs that form the PO team [40]. Other PO roles that have been presented are the feature product owner (FPO) and epic product owner (EPO) [59]. A FPO owns the functionality of a set of features, while EPO is accountable for the unique priority of each feature on the feature backlog.

From the papers about SAP by Schnitter and Mackert [5; 39] they present product teams as a second organizational layer above the scrum teams. A product team is responsible for the work of up to seven development teams and consists of the product owners from those teams. This allows for full engineering coverage of all problem areas expected, well-organized communication among the teams, and direct communication with the development teams to detect and mitigate risks. In the product team they have these roles:

- Chief product owner
- Product team
- Scrum master
- Software architect
- Delivery manager
- Knowledge management and product documentation expert
- User interface designer
- Stakeholder representative

These roles are taken by dedicated people or by development team members. Also at SAP they have an additional, intermediate, layer between product team and scrum teams. These are called area product teams.

The divison of responsibility between the PO team members depend on how the work is divided between the scrum teams. The two main lines of thought are requirement areas with feature teams or component teams [31], as mentioned in section 2.2. In one of the cases presented by Paasivara et al. [31] they divided the APO role between two persons; one system architect and one solution architect. The system architect was a technical person who worked closely with the devleopment teams and communicated with the solution architect. The solution architect was a product management representative with either a business or technical background and did not communicate with the teams, which implies that they only had one PO to interact with. In the other case they had a product ownership team with one PO and ten proxy product owners (PPO). A large feature could have up to three PPOs, while a single PPO might be responsible for a couple of small features. The shared responsibility proved hard and the PPOs eventually had their own features since they all had specific product areas they were most familiar with.

There are many different ways of scaling the PO role, and it is usually accomplished with a hierarchy of POs [58]. The most important criteria to make the PO a success factor is to enable face-to-face communication with the development teams and having a clear prioritization which is efficiently communicated to all POs. It is positive if the PO has a technical background, but it is not necessary. A PO should be good at collaborating, this was mentioned in 2.3.2.

2.3.10 Whole Team Thinking

Ambler proposed that for agility in the large to succeed the team members should share the same philosophical mindset [50]. Whole team thinking can have many names like shared mental models, understanding overall situation, full lifetime cycle or sharing the same philosophical mindset. In section 2.4 we will discuss shared mental models and how they work in both normal and agile teams. Now we will explain why whole team thinking is believed to be an important success factor in large scale agile. In small scale there is only one team and they all work toward the same goals already, which is why it is not that big of a problem. Most of the scaling elements are focusing the attention of all the teams to the whole product instead of ”my part” [40], but this does not imply that this is how it works in real life. Improving the shared understanding of requirements is a key enabler for increased productivity in software projects [60]. One of the reasons for having to look at the bigger picture has to do with dependencies. In a large scale project it is hard not to encounter dependencies to other teams. One of the root factors presented by Martini et al. [41] is that when people discover that interaction is necessary they consider it a low priority and they delay tasks and communication, thus hindering the other team(s) involved. It is therefore necessary for tools creating awareness to help them understand the overall situation of the involved teams. As proposed by Brown et al. [51] it is more important to optimize collaborative teamwork than to optimize individual productivity.

Moore and Spens [49] present several challenges when it comes to whole team thinking. When establishing a consistent culture of working builds there were only a few individuals that were passionate about this, for more information see appendix D.1.5. They hoped that team ownership of code would drive people to solve integration challenges, but sadly it did not. They also pointed out one other problem in large scale, which is that team members must participate in cross-team activities and allocate time to project-wide activities. This could be by the scrum masters having to participate in scrum of scrums, the architects having to gather to discuss architecture etc. In a large scale project it is necessary to do these things to ensure that everything is working according to plan. All the teams therefore struggled with balancing their identity as a team member against their membership in a larger project. This had to do with the demand to participate in these other meetings. The lessons learned by Lindvall et al. [61] is that a part of the project cannot truly be independent, but must interact with and follow the rules of the project overall.

A problem found at SAP [5] was that there had been competition among the team members. Team members felt expected to work harder than was sustainable and they chose more from the backlog than they could implement. This was because the teams saw it more as a competition than a collaboration. They did not think about dependencies or the project as a whole, but only that their team should be the best team. This is one of the reasons that it is important to think about the whole project and not only ”my part” as proposed by Larman and Vodde [40]. It does not help that one team has finished all their work, if there are still tasks left at other teams. The project is not finished until everything is completed. In large scale projects there is a need to sort out the dependencies across the different teams [3]. If this is saved until the end it will only cause more problems. The main cause of communication breakdowns is a ”weak vision of overall goal” [3]. If the dependencies are not communicated the team(s) or team members have a weak vision of the overall goal. Without a shared vision redundant work might be conducted and the system might not fully satisfy the user's requirements [62].

2.4 Whole Team Thinking and Shared Mental Models

This section will introduce the shared mental models theory. Following this is examples of how shared mental models can be applied together with agile practices. In the end we will discuss why whole team thinking is important.

2.4.1 Shared Mental Models Theory

Shared mental models theory is a concept among team members where they are offered a means to explain coordinated performance in teams, especially in conditions where there are a high workload [63]. The theory investigates the role of shared understanding on team performance and has prescribed methods for fostering shared mental models [64]. Shared mental models are thought to provide team members with a common understanding of who is responsible for what task and what the information requirements are. This can allow them to anticipate one another's needs so that they can work in synchronization [63]. Shared mental models is an organizing knowledge structure of the relationships among the task the team is engaged in and how the team members will interact [24].

The term mental model has been used as an explanatory mechanism in a variety of disciplines over the years. Mental models are organized knowledge structures that allow individuals to interact with their environment. They also allow people to predict and explain the behaviour of the world around them. Mental models serve three crucial purposes and that is that they help people to describe, explain, and predict events in their environment [65].

There are two types of mental models: team-related and task-related. The team-related mental models is about how the team is functioning and the expected behaviours of the team [24]. It is the shared understanding related to the other team's interaction and coordination, such as communication patterns, roles and responsibilities of team members, role interdependencies, and background knowledge of each team member [64]. An example of team-related mental models is when a group of students discuss which grade they would like to achieve in a project and how to work together to reach this grade. The task-related mental models contain information regarding the materials necessary for the task or the manner in which equipment is used [24]. It focuses on performing the tasks as a team, such as understanding goals, complexities, challenges, interdependencies, and procedures of accomplishing tasks as a group [64]. An example of task-related mental models is when the students decide between the different equipments and frameworks to use in the project and discuss how this will affect them.

Shared mental models have been used to explain team functioning for years [65]. It facilitates the team's progression toward goal attainment by creating a framework that promotes a common understanding and action [24]. Shared mental models can be used to explain how teams are able to cope with difficult and changing task condition [65]. Teams that share similar mental models communicate more effectively, perform more teamwork behaviours and they generally perform better [24]. Shared mental models theory focuses on the thought processes or activities that occur at a team level and it proposes that effective teams should maintain a shared understanding within a team. Accomplishing team goals without commonality between team members would be nearly impossible [64]. This does not indicate that cooperative or happy teams will always be successful [65]. This only implies that individual members may be headed toward different goals without a shared understanding, which may lead to ineffective feedback or assistance, or the inability to anticipate each other's actions or needs [24]. This implies that having the same goals and being in a cooperative and happy team is not necessarily the same.

There are four specific stages for developing a shared mental model within a team according to Yu and Petter [64]. These four are knowing, learning, understanding and executing. Knowing refers to the stage in shared mental models where team members are exposed to information that is relevant for accomplishing team tasks or projects. When a team is established they will engage in various kinds of information exchange activities, ranging from casual team chat to team building exercise to a formal team meeting. All of these are knowing activities that expose the team members to information that is relevant for accomplishing the team's goals. The team members are in this stage encouraged to share information previously possessed individually by each team member to others in the team [64].

In the learning stage the team members begin to integrate the information they obtained in the knowing stage. The outcome of the learning stage is essential for knowledge construction at the team level. The understanding stage is when the team members develop shared understanding about tasks and the team by solidifying the views and perspectives that each team member holds individually [64]. The goal of this stage is for the team members to reach a consensus and build a common ground so they are ready for the next stage. The executing stage happens when teams develop a shared understanding and start executing their team goals [64].

Shared mental models theory was originally used to account for the performance of military teams, such as whether or not a group accomplished a mission successfully [64]. Stout et al. [63] have explored the relationship between team planning, shared mental models and coordinated team decision making, and performance. The results indicated that effective planning increased the shared mental models among team members. The team members shared an understanding of each other's needs and informational requirements [63]. These studies completed identified that teams that possessed shared mental models were better at predicting other team members' behaviours and needs [64]. It is said that shared mental models have a tendency to shrink with the increasing specialization of team members [22].

2.4.2 Agile Practices and Shared Mental Models Theory

The members in a software team are jointly responsible for the end product and they must develop shared mental models by negotiating shared understandings about both the teamwork and the task [66]. According to Vlietland et al. [59] shared mental models are implemented by grouping people together and stimulate communication and feedback, such as with the scrum of scrums practice. There have been several studies on shared mental models in information systems development performance [64]. Shared mental models among information systems development teams facilitate information sharing between developers, reconciles conflicts between client representatives and software development project leaders. In addition it can improve the overall quality of the software according to some of these studies [64]. By having customer representatives on site they can be part of the development team, which leads to a shared sense of trust, norms and values which could improve the project success. There are different primary goals for the developers and testers in a team, if these can develop shared mental models there could be a greater understanding of the dissimilarities and they can work together to address other issues that could negatively impact the outcome of a software development process [64]. Yu and Petter [64] explained how three agile practices can be seen related to shared mental models.

The system metaphor in agile software development is employed at the beginning of the project to develop a story that makes people understand key concepts of the system and allow them to raise questions. It is used in the extreme programming method, see appendix C.1.1 for more information. This practice encourages the agile team to create an open environment and use metaphors or stories to develop shared understandings regarding system goals, key concepts, major system functionalities, and roles and expertise of the agile team members. The system metaphor is used for the same role as the shared mental model planning practice for the development team's shared mental models. This practice helps the agile team developers to be more effective in the knowing and learning stages, and it allows agile teams to identify unshared information, and quickly communicate with customers in a common language. It improves the team's taskwork mental models in terms of similarity and accuracy in that developers and customers will create a shared and correct understanding of one another's roles, responsibilities, and technical backgrounds. There are two benefits of the system metaphor integrated with shared mental models. The first is that it provides a systematic approach for agile teams to generate system metaphors by ensuring the important aspects relevant to the task and the team are discussed among the users and developers. The second benefit is that the system metaphors enhances the communication effectiveness within the development team by enabling more correct anticipations of one another's informational requirements [64].

Stand-up meetings are used in scrum and are conducted daily. These meetings are short and involve the entire team. Some of the benefits are: daily monitoring and control of the project's progress. The shared mental models in stand-up meetings provide an opportunity to increase team's shared understanding about taskwork through these benefits. It incorporates the shared mental models practices of leader briefings, which is leader communication within teams, and reflexivity. Reflexivity is "the extent to which group members overtly reflect upon the group's objectives, strategies, and processes and adapt them to current or anticipated endogenus or environmental circumstances". The stand-up meeting improves similarity and accuracy of the taskwork and teamwork mental models. The meetings help to identify cognitive differences and goals between scrum master and the team. Through all of the interactions in stand-up meetings team members develop shared understandings of the consequences of the current approach and future steps [64].

The on-site customer in agile practice states that the customer should be present with the development team on a full time basis participating in planning game sessions, acceptance testing, and retrospective sessions. When the on-site customer is looked at from a shared mental models perspective, we can see that it helps the agile teams develop greater taskwork mental models by implicitly implementing the shared mental models practice: cross-training. Cross-training is an instructional strategy in which the team members is trained in the duties of his or her teammates to increase the accuracy and similarity of the team's shared mental models. The team members develop greater taskwork mental models and are able to compensate for teammates' limitations through cross-training. The on-site customer enhances the agile teams' understanding and executing stages, and it builds similar and accurate taskwork mental models for the team. There are two benefits of integrating shared mental models with on-site customer. The first is that it offers a guideline for agile team developers when communicating with the on-site customer. The other is that it offers alternative approaches to learn about the customers' needs [64].

2.4.3 Why is Whole Team Thinking Important?

In a presentation by Moe and Dingsøyr from the Norwegian agile conference "Smidig 2015" they explained why shared understanding is so important [67]. In distributed teams a shared understanding is important for trust. All teams should understand:

- Which agile workprocess do we want?
- What do we wish to accomplish?
- Who knows what?

If the teams do not use the same methods this can be an issue of trust, because they do not trust that the methods of one team works. Having the same goal is also important. If the teams work towards different goals there will be a problem when the different parts are assembled in the end.

Coordination effectiveness is the outcome of a coordination strategy [37], see figure 2.8. The explicit components encompasses the objects involved in the project, while the implicit is concerned with coordination that occurs within work groups without explicit speech or message passing. A definition for coordination effectiveness is:

> *"Coordination effectiveness is a state of coordination wherein the entire agile software development team has a comprehensive understanding of the project goal, the project priorities, what is going on and when, what they as individuals need to do and when, who is doing what, and how each individuals work fits in with other team members work. In addition, every object (thing*

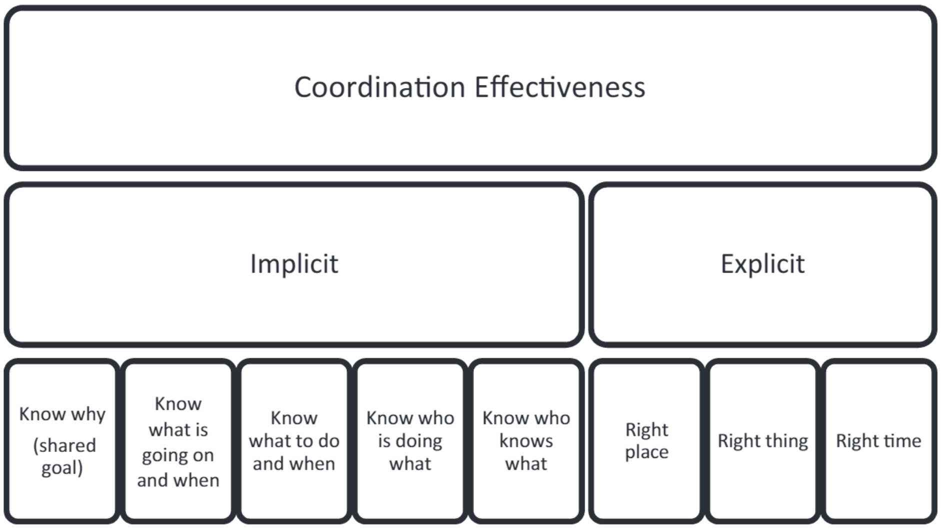

Figure 2.8: Components of Coordination Effectiveness from Strode et al.

Table 2.6: Strode's Implicit Coordination Factors

Component	**Description**
Know why	This implies that each individual working on the project understands the overall project goal and understands how a task contributes to the overall goal
Know what is going on and when	Each individual has an overall idea about the project status, which is tasks currently underway and tasks that need to be performed in the future
Know what do to and when	Each individual who works on the project knows which task they should be working on and when they should be working on that task relative to all of the other tasks that must be completed
Know who is doing what	Each individual knows what tasks others are currently working on
Know who knows what	This addresses expertise location

> *or resource) needed to meet a project goal in the correct time and in a state of readiness for use from the perspective of each individual involved in the project"* [37]

According to Strode et al. [37] there are five factors for implicit efficient coordination which can be seen in the figure and further explained in table 2.6. The five factors are necessary in whole team thinking. Without whole team thinking you risk having several puzzle pieces that do not fit together in the end.

Chapter 3

Method

The method chapter will first give an overview of the literature review before explaining the data collection and data analysis of the case study. This includes explanation of case choice, observations and how the study was completed.

3.1 Literature Review

In this section we will discuss the different parts of a literature review together with its objectives. The selection strategy and research question for this thesis will also be introduced.

3.1.1 Parts and Objectives of a Literature Review

There are two parts of literature review according to Oates [9]. One of them is when a research student explores the literature to look for a suitable research idea and discover relevant material about any possible research topics. This will help the students to get a feel for the area and define a research problem. The other part begins after the topic is chosen and continues until the research is ended or published. Here the aim is to gather and present evidence to support ones claim. Since this also is a case study there are certain elements that a plan should have [68]:

- Objective - what to achieve?
- The case - what is studied?
- Theory - frame of reference
- Research questions - what to know?
- Selection strategy - where to seek data?

To have a successful literature review most of the objectives in table 3.1 must be met. The main objective in this thesis is to see if what we have hypothesized can be accompanied by a result that can confirm it. This does not imply that it is necessary that it is correct in all cases, but at least in some cases.

Table 3.1: Objectives of a Literature Review

Objectives of a Literature Review [9]:
- Show that the researcher is aware of existing work in the chosen topic area
- Place the researcher's work in the context of what has already been published
- Point to strengths, weaknesses, omissions or bias in the previous work
- Identify key issues or crucial questions that are troubling the research community
- Point to gaps that have not previously been identified or adressed by reserachers
- Identify theories that the researcher will test or explore by gathering data from the field
- Suggest theories that might explain data the researcher has gathered from the field
- Identify theories, genres, methods or algorithms that will be incorporated in the development of a computer application
- Identify research methods or strategies that the researcher will use in the research
- Enable subsequent researchers to understand the field and the researcher's work within that field

3.1.2 Selection Strategy and Research Question

For this thesis a supervisor together with a topic was chosen first. After this the supervisor provided several articles on the theme so that it was possible to find a research question. The topic chosen was:

> ***Large scale agile development***

After reading through several of the papers provided by the supervisor it was possible to narrow down the topic even more to:

> ***What are the possible success factors in large scale agile development?***

Since this was still a broad research question it was necessary to reduce it even more, but this was not possible until the success factors had been researched. After finding the possible success factors and analyzed them, the ultimate research question was produced:

> ***Is whole team thinking a possible success factor in large scale agile development?***

After choosing the success factor to focus on, it was necessary to research these more. This then lead to all the search words used for this thesis, see table 3.2. When finding relevant research papers several online databases relevant to IS and computing were searched through, those can be seen in table 3.3. These databases were proposed by the supervisor and by Oates in the book "Researching Information Systems and Computing" [9].

Table 3.2: Search Words Used for this Thesis

Concept	**Keywords**
Agile	agile
Large scale	large scale, large-scale, multiteam system
Success factors	success, factors, tips, recommendation
Whole team thinking	whole team thinking, goal, shared mental models

Table 3.3: Databases Used for this Thesis

Database	**Website**
ISI web of science	http://apps.webofknowledge.com/
IEEE Xplore Digital Library	http://ieeexplore.ieee.org/Xplore/home.jsp
Scopus	http://www.scopus.com/
ACM Digital Library	http://dl.acm.org/

Article Selection

At the beginning the supervisor and other students provided articles based on agile development and large scale agile. After reading through these and proposing a research question the individual study began. The article selection was based on several items:

- Does the article seem relevant from the abstract?
- Does the article have many references?
- Is the article referenced by other papers?

Since there are few articles on large scale agile today, a lot of the papers found were printed out and read through, even though they did not meet all criteria. This was to make sure nothing was missed. Some of the articles that proved irrelevant to the research question could still include theory about the theme and they were therefore looked through briefly. It was necessary to see through the abstract if the article was about an agile development project, preferably large scale, if not the article was probably not relevant for the thesis.

If the abstract was approved the number of references would be checked. The number of references is important especially for a scientific article, because it makes it apparent that there has been research on the theme(s) presented. If there were few references it could imply that it was a lessons learned-article, or based on experiences discovered in the field, which is important since they can say firsthand what worked. These articles were not excluded, but were looked at with a more critical eye. Even though an article did not have many references it did not imply that it was irrelevant, just that it was important to check the facts in other papers. It was therefore checked if the article was referenced by other papers, this was to prove the validity of the article. An article referenced by other papers is often well researched. Checking these two last items was especially important when choosing the final research question after finding the success factors. The success factor chosen could not be mentioned by only one article with few references and no citations.

The reason for this is that it might not actually be a success factor if it is not mentioned by any other articles, and it might not have been researched sufficiently.

In addition to searching for articles, the references in all found articles were checked to see if there were other relevant papers that should be included in this research. From these papers it was again possible to check the references and so forth. This is called snowball sampling [9], when you use the gathered data to find more data. This is useful when it comes to finding persons with similar interests. From this, additional relevant papers were found that contributed to the research question.

Several of the articles found was conference articles and there were few journal articles that mentioned different success factors and lessons learned from large scale agile. This indicates that in some cases there might be the personal opinions of the author(s) or the opinions of some of the people that worked on the projects. Most of the articles were based on one case.

3.2 Case Study

In this section we will discuss the data collection for this thesis, together with choice of case, observations and qualitative data. After this we will discuss the data analysis for this thesis which includes information about the cases and how this study was conducted.

3.2.1 Data Collection

This thesis is a case study where the main goal is to see if there is any truths to the research question. A case study is an empirical method which is purely observational. It can contain different research methods, but interviews and observations are mostly used for data collection in case studies [68]. The idea behind this case study is to find out what is happening, seek new insights and generating ideas and hypothesis for new research, which makes this thesis exploratory. The research process can be characterized as either fixed or flexible. Fixed indicates that it is defined at the launch of the study, while in flexible the key parameters may be changed during the course. Case studies are typically flexible, and that is also the case for this thesis. The primary data source for this thesis is qualitative, see more in 3.2.1.

One of the reasons for choosing an exploratory study is the fact that there have been few studies on the field. This is both when it comes to large scale agile and whole team thinking in large scale agile.

Choice of Case

When conducting a case study the case and the units of analysis should be selected intentionally. The reason that the choice of case study is important could be when someone wants to replicate the study. It can be literally replicated, which indicates that the case is selected to predict similar results, or theoretically replicated where the case is selected

Table 3.4: Taxonomy of Scale by Dingsøyr et al.

Level	**Number of teams**	**Coordination Approaches**
Small scale	1	Coordinating the team can be accomplished using agile practices such as daily meetings, common planning, review and retrospective meetings
Large scale	2-9	Coordination of teams can be achieved in a new forum such as scrum of scrums
Very large scale	10+	Several forums are necessary for coordination, such as multiple scrum of scrums

to predict contrasting results for predictable reasons. When choosing cases for this thesis there were several selection criterias that needed to be fulfilled. These were:

- Agile development project
- Large scale
- Available

When deciding if a project were large scale or not we used the definition given by Dingsøyr et al. [69] which can be seen in table 3.4. From this it is necessary with a case which have from two to more than ten teams. When it comes to the availability criteria this is natural because in practice many cases are based on availability [68]. With the limited time and resources that is this thesis it might not always be possible to find good cases, but it is normally better to have one case than none. The researcher has been lucky enough to have two cases for this study. One of them finished a couple of years ago which have lead to information that others have collected. The other case is an ongoing project where it has been possible to observe several meetings.

Data collection techniques can be divided into three levels [68].

- First degree
- Second degree
- Third degree

First degree implies that the researcher is in direct contact with the subjects and collects data in real time. Second degree is when the researcher collects raw data without actually interacting with the subjects during the data collection. Third degree is independent analysis of work artifacts where already available and sometimes compiled data is used. For the Omega case only the third degree was possible, while for Tellus it was possible with second and third. Second is implemented through observations, and third through observations from other students also observing. It was not possible for this thesis to collect data from first degree.

Observations

”Fieldnotes are gnomic, shorthand, reconstructions of events, observations, and conversations that took place in the field. They are composed well after the fact as inexact notes to oneself and represent simply one of many levels of textualization set off by experience”
-Van Maanen (1988) [70]

According to Wolfinger [70] there are two methods for writing fieldnotes. The first is the salience hierarchy where whatever observations struck most noteworthy is described first. What makes an observation salient is highly subjective and depends upon the particular research context. The second strategy is comprehensive note-taking. Here one does the note-taking from the beginning to the end. One of the advantages with this last strategy is that the researcher often describes events that might otherwise seem to mundane to annotate, but later the data can prove to be valuable.

Observations can be conducted in order to investigate how a certain task is implemented by software engineers. There are two main types of observation: systematic, also called direct, and participant. In the direct observations you decide in advance the particular type of events you want to observe, and use a pre-designed schedule to note their frequency in duration. This can be called a pre-defined system of observations and direct observations is a fairly structured form of data collection. The participant observations is where the researcher takes part in the situation under study and experiences it from the point of view of the others in that setting. This can be completed either by people knowing that you are observing them or not. Here there are several things that are noted and not only pre-defined observations [9].

For this study direct observation is chosen, since it is not possible with participant observation. Participant observation in this case would require training and it is not necessary for the case. Direct observation is primarily a quantitative technique in which the observer is explicitly counting the frequency and/or the intensity of specific behaviours or events or mapping the social composition and action of a particular scene. Since data captured can be observed, and do not require any interaction between observer and those studied, an audio or video recording setup could be enough [71]. For this case the observer is present and without any audio or video recording setup. Some of the reasons for this is the fact that it is not necessarily what is said that is of importance. The observation template is available in appendix B.

Qualitative Data

Qualitative data is all non-numeric data like words, images and sounds. This is the primary type of data generated by case studies. It is possible to do a quantitative analysis on qualitative data, for example by counting number of times a word or phrase occurs [9] Since a case study is a flexible research method qualitative data analysis methods are widely used [68]. The advantage with qualitative data analysis is that it can be rich and detailed and the fact that it is possible with alternative explanations. This indicates that there is possible for different researchers to reach different, but equally valid conclusions.

In this thesis the researcher have gathered data from observations and focus group interviews. In the observations there have been possible with words, images and sounds, while in the focus groups there were only words because they had been conducted by others. There have not been conducted a quantitative analysis by counting words, but words and sentences, and interpretations by researcher in conjunction with these, have been gathered under the different factors.

Principles for Interpretive Field Research

When it comes to the nature of the study it was chosen to do an interpretive view. Interprevitism is:

> *"associated with the philosophical position of idealism, and is used to group together diverse approaches, including social constructionism, phenomenology and hermeneutics; approaches that reject the objectivist view that meaning resides within the world independently of consciousness"*
> Collins 2010[1]

The studies ususally focus on meaning, while in positivism they believe there is a connection. The principles for interpretive field research by Klein and Myers [72] are a set of principles used for interpretive research in information systems research community. See table 3.5 for the seven principles.

Table 3.5: The Researchers Interpretation of the Principles of Klein and Myers

Principle	**Summary**
The fundamental principle of the hermeneutic circle	All understanding is achieved by the equal importance between parts and the whole that they form
The principle of contextualization	The subject of investigation must be understood in both its social and historical context
The principle of interaction between the researcher and subjects	Meaning is produced through the different interactions between the researcher and the participants
The principle of abstraction and generalization	Specific instances of phenomena should be articulated in terms of abstract categories or broader ideas and concepts
The principle of dialogical reasoning	The researcher should make own predjudices and philosophical assumptions explicit
The principle of multiple interpretations	The researcher should explore the varying viewpoints of stakeholders
The principle of suspicion	Sensitivity to possible "biases" and systematic "distortions" in the narratives collected from the participants

[1]http://research-methodology.net/research-philosophy/interpretivism/

3.2.2 Data Analysis

The data analysis phase is where the research is collected and worked through. This is where we begin to compare the findings with the research question. Here we will further explain how we have conducted the data analysis. First we will shortly describe the cases before we explain how this study was conducted. In the end we will display the views of the subject after the researcher presented the results of this thesis in member checking.

The Cases

For this thesis the researcher has been able to follow two cases. The first one is Omega. Omega is a finished project that the supervisor has followed earlier. Therefore I have been granted access to three focus group interviews for this thesis. The codenames is the same as the ones used in other articles. There is one focus group for each of the suppliers and several of the interviewees have been in project management at one point during the project. The background material for the case comes from a Master Thesis from 2015 [73] and an article which is still a work in progress which is about the case [26].

The other case is Tellus, which is a codename given by the researcher since it is a relatively new ongoing project where few others have been included. This case consists of six observations, four conducted by the researcher and two conducted by another student. These consist of: two backlog groomings, two sprint plannings and two retrospectives. Because there has been no interviews the case is based solely on the researcher's interpretations of these observations.

Both of the case studies were conducted in Norwegian since that is the native language of the people involved. This indicates that all quotations used are translated from Norwegian to English. Since the material have been obtained by various methods and in different terms it might not be ideal to compare the different cases. Another fact is that while the Omega-case is finished, the Tellus-case is still ongoing. It is therefore important to state that many of the findings are from the interpretations of the researcher. Others might interpret things differently and might reach other conclusions than the researcher. This is especially important at Tellus where there are only observations and no interaction with the people involved. They might experience the meetings observed in a completely different way from the researcher of this thesis.

Conducting the Study

When exploring the qualitative data the researcher decided to see it with regards of the success factors found in the theory. After reading through the material for each of the two cases the different aspects were put into the different factors that best suited the statements and interpretations of the text.

From the Omega case there was focus group interviews conducted by other researchers. These interviews were first read through and then all the important keywords and statements were collected in its own document. The keywords and statements which were interrelated were put together from this point to simplify the next processes, but each interview were segregated so it was easy to find which keywords and statements which came from what interview.

The observations from Tellus already had keywords and interpretations because they were completed by the researcher and this step was therefore not necessary for that case. After the first step the keywords, statements and interpretations found were put in each factor. This can be called coding of data where parts of the text is given a code representing a certain theme, area or construct [68] which here is the different factors. This was conducted separately for each case and can be compared to tabulation where the coded data is arranged in tables to get an overview of the data [68]. Some items were harder to place and was therefore put in an undecided factor. After all text was undergone by the researcher these were once again attempted to place in one of the existing factors.

This thesis have benefited from several approaches. The immersion approach has a low level of structure and is reliant on intuition and interpretive skills of the researcher [68]. This approach have been used to some degree because the researcher has not had the ability to interview and has relied on observations and interviews conducted by other researchers. The editing approach include codes based on findings of the researcher, which is the different factors found earlier. These have been used when reviewing the two cases and compared them to each other. The last approach used is the template approach where the research is based on research questions [68]. The quasi-statistical approach, where there can be a calculation of frequencies of words and phrases, did not seem appropriate in this thesis and has therefore not been utilized.

Member Checking

After the observations and analysis of these were concluded the researcher presented their findings to the Tellus-project. The people at Tellus did not know the purpose of the project and therefore they first needed an overview of the study. After this the researcher presented the findings from the case and compared it to the Omega case. In the end further actions for the project were presented.

The people involved agreed with what the researcher highlighted and they had already made several changes since the researcher had observed them. They seemed to appreciate the feedback given and were interested in how to begin new projects to try to eliminate some of the problems earlier in the process.

Member checking can bee seen together with the third principle of Klein and Myers [72]. This principle is about the interaction between researcher and subject. The researcher did not communicate with the subjects during the observations, but later there were interaction where it was possible to find unity between researcher and subjects.

Chapter 4

Results

The results chapter contains the results from the two case studies conducted. First we will give an overview of the two cases. After this we will present the results from each case in conjunction with the different success factors.

4.1 The Cases

For this thesis the researcher was able to follow two different cases; Omega and Tellus. Both of them have been anonymized. For more information about the data analysis see 3.2.2. Here the cases are explained with a basis of the project and an overview of the work areas. For an overview of both projects see table 4.1.

Table 4.1: An Overview of the Cases

Project	**Omega**	**Tellus**
Budget	140 million euro	30 million NOK, exclusive VAT
Duration	January 2008 to March 2012	Fall 2015 to early 2017
Number of people	175	15
Number of teams	13 at most	2 squads in one team
Location	Co-located	Distributed
Time of sprints	3 weeks	4 weeks
Different companies	5	1
Releases	12	About 10
Tools	Jira	Jira
Information gathering	Focus groups	Meeting observations
Conducted by	Others	The researcher and other
Status	Finished	Ongoing

4.1.1 Omega

This section is based on the information from Andreassen [73] and Dingsøyr et al. [26]. Omega was a project both initiated and completed by Gamma, a public sector department in Norway. In Gamma it was necessary with a new office automation system and the main arguments for initiating the programme were public reform, and because the current platform was outdated. The reason for choosing the agile development methodology was because little was known about the public reform at the beginning of the project. This indicates that there were high levels of uncertainties.

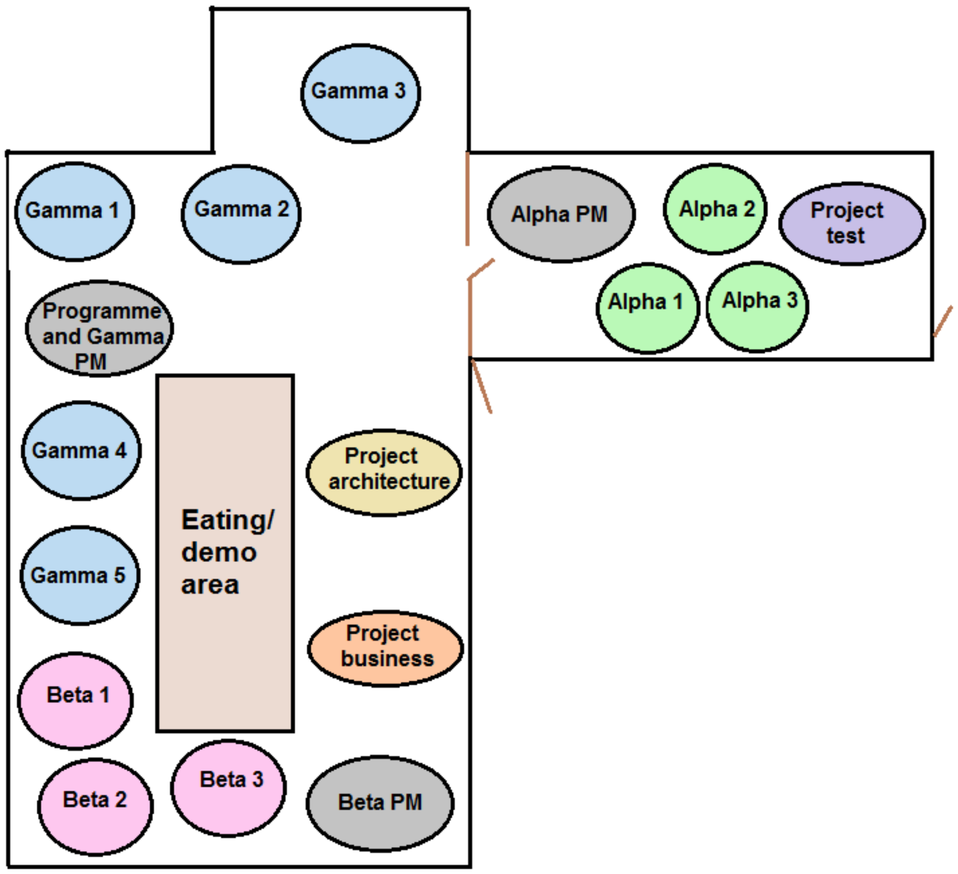

Figure 4.1: An Overview of the Open Work Area at Omega

A little way into the project everyone was placed in the same open work area, see figure 4.1. The projects were performed across the three organizations: Alpha, Beta and Gamma. At most they were up to thirteen teams placed on different tables in the open work area, together with tables from project managements, business, architecture and test. The scrum teams mainly consisted of about eight to ten members with different roles. In addition to given roles, the members were cross-functional so that they could be e.g.: sixty percent tester, thirty percent developer and ten percent designer.

It is important to note that the application was recreated from an earlier version only with improvements. The outcome and the complexity would have been entirely different with a completely new application. Omega is one of the largest IT development projects in Norway to date and is considered a success.

4.1.2 Tellus

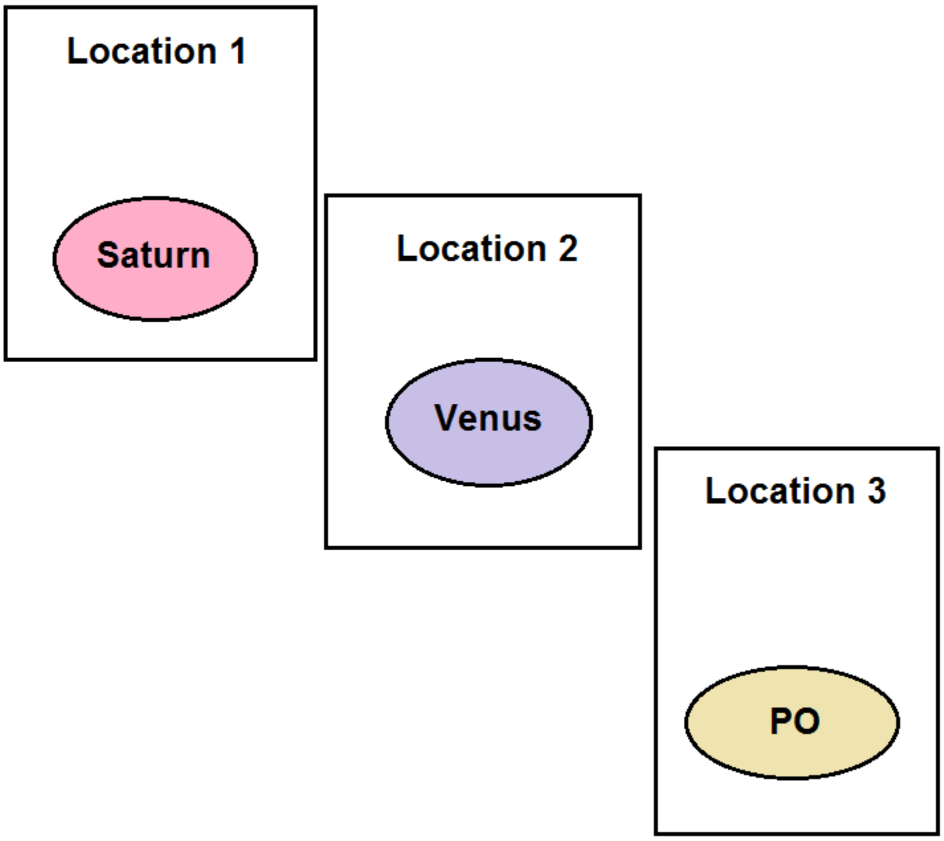

Figure 4.2: An Overview of Tellus

The Tellus project is a control- and supervision system with an iPad-application and web-based request tracking-system in the Jupiter department. The competition for the project was big and in the end the company Mars won the contract. Mars is a small company based in at least two cities in Norway, and among previous clients we can find banking in addition to several public departments. The Tellus project is the biggest individual project in the history of the company. The project is still ongoing and is supposed to last for about one and a half year with parts of the solution operational later in 2016, and fully operational in late 2017.

The solution will gather information from underlying registry so that workers in the Jupiter department do not have to search many systems before supervision. During supervision the iPad-application will be used to document findings. It is a requirement that this solution should be able to synchronize offline. One of the reasons Mars won was because of their choice in reusing code. This involves Mars reusing code from other deliveries from earlier and it opens up for other departments possibilities to take use of the system at a later time.

Mars chose to use agile development methodology for this project because of the requirement of having short time to market since the department should be friendly to changeovers. The Tellus project is conducted over one team which is divided into two squads located in different cities. The customers and product owners is again located in another city, see figure 4.2. This is some of what makes this project distributed[1].

4.2 Success Factors

In this section we will present the findings from the two cases; Tellus and Omega. We will see both in conjunction with the factors and compared to each other.

4.2.1 Agile Software Engineering Techniques and Agile Evangelist

Both the projects needed to follow agile engineering techniques because of the changing requirements of customer. For Omega they knew little of the public reform, while for Tellus it was necessary to have short time to market, and parts of the delivery operational during project duration.

Both projects used the scrum methodology with sprints. Omega had three-week sprints, while Tellus had four-week sprints. Tellus noticed the problems with these long sprints because they did not manage to plan that far ahead.

> *"It is hard to plan a whole month ahead since it does not get correctly synchronized with Jira."*
> Developer at Tellus

They eventually decided to divide the sprint into two parts; a and b. Another change they made was instead of detailing the tasks together, each developer got to detail their own tasks. This was to make sure they did not have to work on several tasks that belonged together, but only one task at a time. At the Omega-project they did some scrumban/kanban which meant that they needed to finish one task before starting another.

[1]Sources: official pages of the Jupiter department, regjeringen.no, digi.no, kystogfjord.no

At Omega they had agile evangelists and people were willing to do an effort to be agile. They took courses to learn the techniques and everyone was eager to soak up more knowledge. This was because there were few who had had agile experience before entering the project. For the Tellus-project it is not known whether they were coursed during the project, but they saw that they did not do the backlog grooming according to the book during one of the observations. This did not only apply to the team, but to the product owners who were not good at prioritizing.

Omega were big users of pair programming in their project. Beta introduced it, and soon all teams were doing it. The goal was for everyone to have programmed with everyone so that it was possible to create a line between everyone in a team. This was a hit which ensured everyone on the team aquired knowledge from everyone, and that the coding resembeled. They therefore tried pair programming across teams, but that was not a success. The pair programming was good when bringing new people into the project, but later in the project several people startet programming alone.

4.2.2 Responding to Change

When it came to responding to change this was easier for the researcher to notice in the Tellus-project since that was still ongoing, while Omega was finished. From what the researcher observed they were good at taking action when necessary. The case that all meetings observed were completed in different ways each time display that they were able to see the need for change. They also decided on a new way of dividing up the tasks in a sprint. This was eventually up to the developer working on it. This was a way of making sure that there were not several subtasks that needed to be implemented simultaneously which could lead to a developer having several unfinished tasks. In addition they decided to divide the sprint into two parts when they saw difficulties with planning for four weeks.

Whole team thinking can also be found in this factor. After dividing the sprint into two parts because of the planning difficulties at Tellus they decided to have a common burndownchart for both squads in each sprint. At Omega they made sure to change the arenas when they saw that some arenas did not work, more on that can be seen in section 4.2.5.

4.2.3 The Team

The team members at the Omega-project did not necessarily have much agile experience. Because of this they might not have been so determined when it came to self-organizing in the team, because they from the start knew that the project involved several teams working together. They did have people who were interested in learning about agile, which implied that they should learn while doing, and then customize to their project, and not agile projects in general. It is not known how much agile experience there is in the Tellus-project, but with the increase in agile methodology it is likely that some have experience. At Tellus they prefer to look at themselves as one team with two squads rather than two teams. This helps with the whole team thinking as well.

At Omega they mentioned the disadvantage with big projects which is that there are always people coming and going. Instead of creating new teams with the new people they made sure to separate those experienced so that everyone could be on a team with someone experienced with the project. The teams were more specialized in the beginning of the project. At Tellus it was the locations that decided where people were put. In addition they saw a problem with people coming and going, even though their project was smaller than the Omega case.

At Tellus they had a problem with not having a good feeling in the project and low achievement in sprints. This lead to lack of ownership in the project.

> *"I really feel like we all want it so bad, but we are more or less frustrated by the fact that things are moving slow, and we feel like we are still at the beginning after five months."*
> Developer at Tellus

They all saw that they had potential, but they were struggling with acknowledging the results for themselves. At Omega their biggest problem were the fact that not everyone was happy when it was necessary to rotate the teams. The teams were very self-organizing and the people felt more like they were working at Gamma because of the time spent there compared to where they originally came from.

Omega had many different arenas for people with certain roles. This is because of the size of the project. This can be compared to whole team thinking because people needed to spend time outside of their team to ensure that the different pieces of the project could fit together. For Tellus, a much smaller project, this was not observed, but probably it did not take much time from other duties.

4.2.4 Leadership

The leadership in the two projects were very different. At Omega there was more of a hierarchy because of the size of the project, while it was not that visible from only meeting observations at Tellus. In the meetings at Tellus there were someone that usually took control, but this control was sometimes divided between several persons. They normally did not have a structured plan of what they should go through and they did not have a secretary, or someone to write down important aspects that came up during the meetings. This was however implemented in one of the last meetings observed, by someone making a list of what everyone should do until next time.

At Omega they had project management teams and different kinds of leaders like scrum master and others. Even though project management was not in the teams they were visible and showed interest.

> *"Just management by walking around, talking around."*
> Project management at Alpha

They visited the teams at least once a day and had the opportunity of watching the boards, which made it easy to see if there were tasks that they struggled with.

The leaders at Omega were also responsible for making sure that the teams did not only care for themselves. This was illustrated by the leaders trying to encourage everyone to help each other before being asked to help. They often knew if someone was working on something they had knowledge about. This was one of the methods the leaders used to ensure whole team thinking in the project. It was much easier for the leaders to see the whole project which is why they were responsible for providing the way of thinking down to the teams.

4.2.5 Bridgehead

A bridgehead is a way of coordinating what happens at different teams in a project. For the Tellus-project a lot of the meetings are conducted together with both squads and sometimes together with the product owner group, all known arenas can be seen in table 4.2. At the Omega-project they had several different types of arenas to communicate the project to the others, all can be seen in table 4.3 [26]. These arenas helped coordinate the project and ensure that the project in the end was one product and not several different pieces. The arenas were a way of securing the whole team thinking through the whole project.

Table 4.2: The Different Arenas in Tellus

Arena	Description
Demo	Demonstration of developed user stories where everyone could participate
Backlog grooming	Completed together with product owner and important people from both Saturn and Venus
Sprint planning	These have been conducted both in the squads, with both squads and with both squads together with the product owner
Retrospectives	Have been completed together with both squads
Tech talks	Weekly between the teams
Wiki	Different documentations like architectural guidelines, team routines and retrospective reports were documented in the project wiki
Jira	Tool for managing the backlog and setting tasks and see burn-downcharts
Live-feed	Both squads have live-feed of the other to feel closer to each other

At Omega the arenas changed over time after what worked best. They were user-controlled, which implies that when they did not find an arena necessary they ended it, which opened up for other arenas. Since Tellus is still an ongoing project it is not easy to know how things will develop later. From what was seen in the observations they changed

Table 4.3: The Different Arenas in Omega

Arena	**Description**
Demo	Demonstration of developed user stories where everyone could participate
Experience forum	A forum at subcontractor Alpha for scrum masters, development manager and agile coach focusing on development method
Instant messaging	Used for open technical questions and social activities like wine lottery
Lunch seminar	Seminar with two or three short presentations during lunch on topics like architectural components, project management or on how to follow up a team
Metascrum	Two meetings per week with project managers from development, architecture, test and business projects, as well as subproject managers from the development projects
Open space technology	Process where all participants suggested topics for discussion where participants were free to join discussion groups of interest
Retrospectives	Used mainly on team level, but also on subproject level at times. All were documented in wiki so everyone could see
Scrum of scrums	All three subprojects had scrum of scrums with their teams two or three times a week. Scrum masters and project managers attended, and sometimes others like product owners and test managers
Technical corner	Architectural briefings in the subprojects at Beta, where team architects briefed the team
Wiki	Different documentations like architectural guidelines, team routines and retrospective reports were documented in the project wiki
Rotation of team members	Members were sometimes rotated between teams
Open Work area	The project with all teams was situated in an open-plan office space on one floor
Masterplan	The programme established a common product backlog as a master plan, with 2 500 user stories organized in 300 epics
Stand-up	The time of stand-up was set so it was possible to visit others teams' stand-up

their arenas continously. The first sprint planning was just in the squad, while the other were divided into three parts with one part with product owner, one within each squad and one with the whole team, in other words with both squads present.

At Omega they saw a shift from formal arenas into informal arenas:

> *"I imagine that those arenas are more important at the beginning, but it becomes less important when you get to know each other and get used to going over to talk to the peson you know can help you."*
> Person from Beta

These informal or unofficial forums like joint coffee-breaks etc. appeared to become automatic after a while. At Tellus on the other hand, it is not easy having informal arenas since they are situated in different locations.

There were some problems at both projects when it came to some types of arenas. At Tellus they discussed the fact that the retrospectives should be conducted face to face, rather than over video. This was to be able to be closer to each other. At Omega they noticed that it was hard to get everyone on the same level in the metascrum. Some people used more time than the ten minutes which were intended for each, and detailed more than necessary. This resulted in some waisted time since it was not all detailed information which was necessary for everyone.

4.2.6 Co-Location

Both of these two projects chose entirely different when it came to where the teams were placed. At Omega they chose to co-locate everyone in an open work space area. This included project management and product owner, or functional experts, which were placed in the same room. In Tellus they chose a completely different practice and worked distributed within the teams, or squads as they called them. They were not only in different locations but also in different cities. This included the product owners which did not sit with any of the squads, but in another city. From the time they were all put at the same location at Omega, everyone was one hundre percent with their team, while at Tellus they had one team member that did not sit with everyone else.

Because of the distance between the teams in Tellus they have put up a video live-feed over the offices so that they can see each other all day long. At one point during the project one squad, Saturn, lost the others feed. While they were interested in getting it back, the other squad, Venus, did not really see the point because they did not feel like they were dependent on the other squad.

> *"I think that Saturn is more concerned about it than Venus is."*
> Developer at Venus

> *"That is because the arrow of dependencies goes that way [Saturn dependent on Venus]."*
> Developer at Venus

In the end they agreed to fix the TV.

At Omega the fact that they were all located in the same room was important. This was because it opened up for more informal communication arenas, and it made the work easier for mangement. When they saw a team struggling they could just go up and help right away, instead of not knowing about the struggle.

> *"That with being in the same open space I think was an important factor here. This is something I notice now at Zeta, where we do not have this one space."*
> Project management at Alpha

In the Tellus case they saw the necessity to work together at the same location, especially when working on tasks with dependencies to the other squad.

> *"I feel that all I have done these last two weeks are coordinating[...]. I feel that I got a lot more from the ten to fifteen minutes where we booked a meeting room than I have gotten [from other platforms]."*
> Developer at Saturn

They discussed the fact that it might be necessary for more traveling between locations. The only problem is that it requires money from the budget so it is important to decide what to prioritize.

Both projects saw the fact that it was much easier when having somebody to help. To aquire good answers to questions, when possible it should be conducted face to face, rather than sending mails or using other tools.

> *"What is important in agile, is that you are to deliver something in three weeks. Then you do not have the time to wait for someone to read all their mail before getting to the one you sent two weeks ago, because that is at the bottom of the list. You have to go there and ask for attention. You might not get the answer in ten seconds, but for him to open the mail, read it and answer, that takes time."*
> Project management at Alpha

It takes a lot less time when getting hold of the person and asking the question, than waiting for an answer which may never come.

When it comes to whole team thinking in this factor the fact that Tellus had a live-feed ensured that it was easier to collaborate because they were able to see if the person they needed to be in contact with were available. For Omega this could be accomplished by checking the open work space. This was less time-consuming than having to wait for answers on chat or mail which ensures that everyone is more effective in the project.

4.2.7 Product Owner

There is a big difference in the product owner at both Tellus and Omega. At Omega the product owner was active and used its best people.

> *"It is really a success criteria - take the best people and give them authority and responsibility."*
> Project management at Alpha

The customer used their best business architects in the project and they were put in the same open work space ninety-five percent of their time. In eigth out of ten situations they were able to take a decision then and there. In Tellus at the other hand the customer is situated in another city than the team. This complicates things when there are questions that require answers.

At Tellus they are struggling with the product owner not knowing what it entails when being a part of an agile project. They do not have the information ready, they have to be constantly reminded of what they should do and they do not prioritize. Since Gamma is a part of the development in Omega, and have been involved in choosing the agile development methodology they seem to have had an understanding of what is required by the customer.

There were three levels of product owners in Omega. There were one at the top, one for each of the suppliers, and functional experts.

> *"My experience of our product owner was that he was more of an administrator, that was formally the product owner role, but the functional experts were essential since they had a strong connection to the project, and that I think was crucial for the project going that well."*
> Developer at Gamma

It was important in a project that big to have several product owners and they were good to come up with defined tasks. At Tellus they had a product owner group with people responsible for the different parts of the project.

According to the observations at Tellus they were happy with their product owner and felt that they appeared to be open and trusting. They were also good at answering when it came to technical issues. The problem was that it did not seem like they had the authority to make snap decisions, or that they talk to each other. In some meetings they disagreed with each other, and these are things they should have dealt with before, and not during the meeting. It was apparent that the product owner could be somewhat unclear and was missing presence.

> *"Overall they are reactive and not proactive. We need to give them concrete tasks constantly and then we need to remind them a couple of times before it goes quiet."*
> Developer at Tellus

The team struggled with having to wait for the requirements specification before conducting some of the developement tasks.

Both of the projects have several POs, which can make it difficult when they do not think about whole team thinking. At Tellus they did not seem to talk together which implies that they did not display a united front to the teams. Since they did not have a united front they also struggled with prioritizing, which made it harder for the team to know what to produce when. The hierarchy at Omega ensured that there were always someone with the knowledge and authority which indicates that it was less time-consuming because they were all aware of the whole project in the end.

4.2.8 Whole Team Thinking

Both of the projects had several teams or squads. In both the observations and in the focus groups they spoke as though the teams were separated in ”us” versus ”them”. In Omega the teams experienced great team spirit with team names, effects, cheers etc. It was obvious that everyone was more committed to their team and not the project itself. The priorities was to do things for the team first, and if there was more time, that could be used to help other teams.

When it came to the burndowncharts Tellus eventually decided to have a joint burndownchart per mini-sprint. It then appears that they had had separate charts earlier. At Omega they had burndowns per team which might have had a connection to the competition between the teams. When they noticed this they removed the visible burndowns.

> *”We tried to run the focus to the fact that we are one provider, and what we deliver we deliver together.”*
> Project management at Beta

They saw that how much each team delivered did not matter as long as the cooperation of the total delivery was good. This was probably one of the reasons for Tellus to choose a joint burndownchart.

When there was a problem in Omega because of dependencies between the teams people were quick to judge the other teams. This was resolved with someone from each team being put together to solve the problem together. This helped with the fact that people could no longer blame each other. Omega was also proactive when it came to dependencies and had regular dependency-meetings. At Tellus they were not that proactive when it came to dependencies. They lacked structure in both planning and meetings, which led to them not always being able to go through all tasks and see if there are any dependencies. It was simply not prioritized enough to find the dependencies so they could discuss with the other squad. When both squads were together in the meeting they did not spend much time on discussing the dependencies they had found, if they had found any at all. In addition it was apparent that Venus believed that it was only Saturn who had use of the video live-feed between the offices.

Another thing at Omega was that the teams that did not depend on other teams, did not bother to deal with anyone else. There were only the teams with common dependencies, or working on the same things, which communicated with each other.

> *”The way everything is build and stitched together makes it painful when working together so they simply did not have any other choice than to talk*

> *together to find a solution."*
> Team architect at Gamma

At Tellus they saw the same problems and the need to talk together when working on the same problems. There they also saw the necessity of working closer together, in other words being in the same location when working on the same objects.

One of the biggest problems with being able to do whole team thinking was the contracts. For everyone it was more important to deliver what was in the contract, than thinking of the project as a whole. They did however see that there is no winner, either everyone do well, or everyone do poorly.

> *"Seeing yourself in a bigger picture. We struggled there..."*
> Project management at Beta

For Tellus which have only one contract together this is different. They had selected to do the project distributed, but they still remained one team. Here they just had to be better at not only saying that they are one team, but also think it. The different squads usually use the words "us" and "them" which inhibits their actual meaning.

At Omega there was said that if a team saw that they were able to finish their tasks, they should speak up and take some tasks from those not so lucky. This was to better help each other out. Another item which was brought up was for everyone to be more proactive when it came to helping each other. This applied particularly when they knew that someone would require their help. In most cases it appeared that everyone was more reactive and only helped after being asked. The management did do their best to get everyone to think more for the whole project than only their own part. For Tellus it was obvious that later in the sprints the Saturn office did not have as much tasks as Venus. This was discovered early and Venus were more than happy to provide Saturn with more tasks. The division of tasks or main components divided earlier could have been somewhat premature and it might be something that should be implemented more frequently.

Chapter 5

Discussion

In the discussion chapter the findings from the results chapter will be discussed. We will discuss these results according to the research question and together with Strode's implicit coordination factors. In the end we will evaluate this study both according to Klein and Myers and if the cases were agile.

5.1 Research Question and Success Factors

In this section we will first present the research question before discussing the results from the success factors in regards to the research question. In the end we will give a brief summary of the findings.

5.1.1 Research Question

Now that we have seen some of the results from the two cases it is important to see it in regards to the research question and the theory. The original research question was this:

> ***What are the possible success factors in large scale agile development?***

Since this subject would be to broad for a master thesis it has been narrowed down to this:

> ***Is whole team thinking a possible success factor in large scale agile development?***

The discussion here will primarily revolve around the research question and the results from the observations and interviews for the two different cases; Omega and Tellus.

If we were to explore the other possible success factors we found in the two cases, mentioned in chapter 4 there would be necessary to explore these fields further. This is why we will see them together with the whole team thinking factor. Several of the factors can be seen in regards to whole team thinking.

5.1.2 Responding to Change

When it comes to responding to change we can see from both cases that they at times saw that it was necessary to focus on the whole team and not just the teams separately. At Tellus they decided to have a common burndownchart instead of one per squad, this can be seen as a first-order change [34]. They are after all working toward the same goal. This was not the only thing they decided to have common for the teams:

> *"Then we agreed to make one overall picture. At least to describe the most important components and how they interrelate."*
> Meeting leader at Tellus

At Omega they tried to encourage everyone to help each other and they found new arenas where they could collaborate. When they saw that the need for one arena disappeared, they let it go instead of insisting on using people's time when they not did believe it was the right way to go, this can be seen as a second-order change [34]. They saw what the requirements was for the teams to collaborate and used it to improve at all times for the project to go as smoothly as possible.

5.1.3 The Team

In section 2.3.5 we read that Moore and Spens [49] concluded with that it is not necessary to base the hiring of people on agile experience when working in large scale projects. This is because people who are used to agile might not thrive in a large scale environment. At Omega there were several people who were not acquianted with agile methodology. This probably had something to do with the fact that the project was started in 2008, but still it showed that it is not necessary with agile experience. Everyone at Omega were eager to learn about agile and they spent time to improve themselves. The reason that it might not be necessary with agile experience in large scale projects is because it is more important that people remember they are only one of several teams working on the same project.

With a growing number of teams and people it is necessary to spend more time outside the teams, also proposed by Moore and Spens [49]. This is to make sure that everything in the project is going according to plan. Without these extra meetings there are several questions that never will be answered. When observing at Tellus it was obvious that several of the members needed to spend much more time in meetings than with their squad. This was something mentioned at Omega where there was even more necessary to spend time outside the teams. This could be seen by this quote from Alpha:

> *"As an employee at Alpha I had in longer periods no feeling of being an Alpha-employee. This was where I was. And in periods - both here and in other projects - where there is so much overtime that you cannot attend Alpha-arrangements you do feel more like a Gamma-employee."*
> Team architect at Alpha

It was necessary to spend time outside the team to get a better overview of the project. It was why people felt more at home at Gamma since they spent all their time there. This is something that probably made everyone feel more comfortable with each other and not thinking so much about the fact that they were from several suppliers since they all worked at the same place.

5.1.4 Leadership

Leadership was important when recognizing the importance of the teams to think as one unit and not as separate parts in the project. This can be seen by one of the behaviours presented by Moore: driving both team and project success [52]. Without good leaders who acknowledge the problems and make it their mission to handle the problems, it is not possible for the teams to understand they are only one part of the project. The leaders are the ones who are able to see the project as a whole, which makes it necessary for them to communicate this to their teams.

> *"You want people to be able to handle it at the lowest level possible. But maybe it is necessary to communicate more with regards to project management that this is one project, and the important items are this and that, and get rid of the mental blockage that we first need to think about ourselves and then maybe help or support others, or ask others for help."*
> Project management at Beta

At Omega they tried to remind everyone that it was necessary to help each other. They also made sure to prioritize and explain when other teams' tasks was important to prioritize over others. If they had not done that the people in less prioritized teams would have been displeased. At Tellus they did not seem to have very clear leadership in meetings, but they were good at involving everyone which caused them to more conveniently see the bigger picture.

5.1.5 Bridgehead

The bridgehead is necessary so the teams know where they all are at, also called coordinating [41]. They are all supposed to deliver something together, and if they are not coordinated, this would be extremely hard. At Omega they had several arenas for discussing parts of the project and updating each other where they all were at. An example are the scrum of scrums which Paasivara et al. [56] concluded with worked poorly, but in this project worked pretty good [26]. They had meetings where they mapped the dependencies so everyone knew which teams they were required to have contact with.

> *"It has been controlled by what people wanted to discuss, in the end there were no more themes requested and it was dropped. The project management did not invent several themes just to keep it. They were kind of user-controlled."*
> Team architect at Alpha

At Tellus they had joint meetings for the same reasons as at Omega. This made it possible to see early if they were dependent on each other in cases and who should cooperate more. Without all these arenas the projects could face issues with a failed final product with parts that could not communicate. It is necessary for communication and collaboration between all teams, whole team thinking, when there are several teams and people involved.

5.1.6 Co-Location

Without co-location, people always have to consider that they are not the only ones working on the project and understand the need for cooperating. This is necessary even when you do not see the ones you have to cooperate with. At Tellus this is an important aspect and it is why they put up a video live-feed. This simplifies the fact that you can check if the one you require is actually available or not. At Omega they saw that when everyone were acquainted they did not have a problem with going to the person and ask when necessary instead of sending a mail and waiting for an answer. It was easier to go outside their own teams. This is one of the aspects when you see that people understand the fact that they are not alone on the project.

5.1.7 Product Owner

In large scale it is necessary to scale up the product owner because it is difficult for one person to be responsible for several teams [31]. It is difficult to find one product owner who is familiar with every aspect of the project. At Tellus they had a main (chief) product owner and a product owner group, just like at SAP [5]. This was necessary to ensure that all necessary parts of the project were preserved. At Omega they had a hierarchy of product owners which achieved the fact that every team had access to persons with authority and responsibility. This ensured that everyone was able to make a satisfactory product in the end.

> *"It is really a success criteria - take the best people and give them authority and responsibility."*
> Project management at Alpha

If the product owners did not prioritize they would have made it more difficult for the teams to choose which tasks is important. This again could result in a failed solution. If the product owner does not think about the bigger picture, it is hard for the supplier to think of the bigger picture and the fact that they are all one team. This can be noticed if one in the product owner group believes that their part of the product is the most important one.

5.1.8 Whole Team Thinking

As we have seen from the two cases whole team thinking is a subject which is often brought up. This does not indicate that it is presented in those words but we can read it between the lines.

> *"If one can help someone else deliver something more important, that would be better. Maybe even if we could help someone at Alpha or Gamma to deliver something which is more important, that would actually be better. Seeing yourself in a bigger picture. We struggled there... We did not do anything about it and understood that for the teams it was more important to deliver what was written in the contract."*
> Project management at Beta

This can also be viewed by how they talk about the different teams. There is much mentioning of ”us” versus ”them”, and the way the teams distinguish themselves from each other. Because the teams are measured individually instead of together it is necessary for each team to think of themselves, or their own part as proposed by Larman and Vodde [40]. When the burndowncharts and other measuring components were removed at Omega, and were no longer visible, it was easier for the teams to not think of the others as competition. This is probably why Tellus chose to have a joint burndownchart. They are after all working toward the same goal and if one team delivers and the other does not it is still the project in whole that suffers.

Whole team thinking, or shared mental models, is when someone work together with the same goals or philosophical mindsets [50]. In one team this is easier to accomplish than with several teams working on the same project. The different teams all have their own goals and work toward them. The problem is not that they have their own goals, the problem is when they forget that they are not the only ones in the project. It is therefore necessary to remember that what they are working on probably would have to compile with what the other teams do.

The dependency problem is a big one when it comes to several teams on the same project. As we have read from the theory, somtimes people just downgrade the problems and put them on hold, just like in the paper by Martini et al. [41]. The problem then is that the problems increase. From Omega we could see that people started blaming each other for the problems. This is why they put the people from the different teams together to try to resolve the problems together. When they did work together they did not have anyone other than themselves to blame.

> *”There is one thing I remember where we had the most problems finishing something, there was a lot of things going on and in the end we just needed to take one developer from each team and put them together [...]. In other words finding the entirety together.”*
> Head of development at Alpha

At Tellus they did not appear to be that proactive when it came to the dependencies, which is why they might have some problems delivering in the sprints. There is a big difference in how Omega and Tellus handled the dependencies, and it shows that with a bigger project they are better at being proactive, because it is that much harder when they are not aware at the beginning. At Tellus on the other hand they are only two squads, which may be the reason why they do not spend more time. This is probably because they belive that when the dependencies arise, they will not be that big of a problem.

The division of tasks in large scale is difficult, not only because there is no way of knowing exactly how much job every task will require, but also because of the level of competence in the different teams. At Omega they tried to solve this by having the teams that knew they were able to finish their tasks to take some from those who did not think they would finish. Since Tellus is of much smaller scale they can divide the tasks more when they see the need for it because it is easier for them to have a complete overview of project and tasks.

> *"Saturn struggles in 7b with few tasks and needs more work, but if you only generate four days there is not much point if it leads to more work for you."*
> Saturn developer

Because of the better overview at Tellus it is therefore easier to discuss further work together.

As presented earlier we can see from the way everyone talks that there is a division between the teams. At Omega this also applies to the different suppliers in the project, while at Tellus it applies to the different locations. In both the observation notes at Tellus and in the focus group interviews at Omega there is much "us" versus "them". At Omega where the teams had their own names, cheers and people were resistant to switch teams this was easy to see. This kind of talk might be something of what separates the teams in the project from each other. This is especially a problem since it comes from project management, which is the case for several of the people in the focus groups from Omega. At Tellus we could see that they do their best to be one team with two squads, while at Omega they saw themselves that it was necessary to explain that they are all working on the same project. This was to ensure the fostering of shared mental models [64]. These are ideas that help the shared mental model overall, or as it is called here, the whole team thinking.

5.1.9 Summary

What can be seen from all this is that there are definitely some things that could be defined as success factors when it comes to large scale agile development. In addition we can see from the two different cases that there are several differences in how they approached these possible success factors. When it comes to location Omega is co-located while Tellus is distributed, which is the most prominent difference between the two in addition to the size. Other differences are the teams; where Omega had several teams, while Tellus had one team which consisted of two squads, and the product owners where Omega had a hierarchy with functional experts with authority, and Tellus had a product owner group where they did not appear to always have the authority. We can see clearly that some of the possible success factors are specifically mentioned both in observations and focus groups without asking for it in particular. All found success factors for the cases are listed in table 5.1. NF implies that the factor is not found while S implies that it is found in some degree, but not so much that it is presented here.

We have discovered that several of the possible success factors can be seen as a remedy for whole team thinking and shared mental models. We can link the different success factors together by seeing what they have in common. As we can see from table 5.1 not all factors are found in both or any of the cases. Since there was observations at Tellus, it was not possible to ask about the different factors. At Omega the researcher was not involved, which is why it was not possible to ask for more information about them.

It would be hard trying to replicate these two cases for research. This has to do with the fact that no project is the same and that there always will be a difference in the knowledge

Table 5.1: Success Factors Applicable Found in the Cases: Y= found, S= found in some degree, NF = not found

Success factor	**Omega**	**Tellus**
Customer collaboration	S	NF
Agile software engineering techniques and agile evangelist	Y	Y
Responding to change	Y	Y
Team	Y	Y
Leadership	Y	S
Planning	Y	Y
Testing	Y	S
Governance	NF	NF
Divide after you conquer	NF	NF
Continous integration	NF	NF
Bridgehead	Y	Y
Co-location	Y	Y
Product owner	Y	Y
Whole team thinking	Y	Y

and capacity of team members. Another is the fact that there were certain limitations like the time limit, restriction of resources for the two cases, and the fact that this is mostly based on the interpretations of the researcher. When it comes to Omega this was also one of the first large scale agile projects, and not everyone were that familiar with agile as they might be now.

> *"We have recreated a lot of the same later. But there is a discussion of how successfull it has been. There is also a matter of the governing conditions and settings where you try to recreate it. We have discussed the ability that people can take decisions on the spot, this is not possible everywhere."*
> Head of development at Alpha

Strode and Whole Team Thinking

In the theory chapter, chapter 2, we explained why whole team thinking is important and we looked more closely on Strode's components for implicit coordination effectiveness, see 2.4.3. In table 5.2 and 5.3 we have the findings from Omega and Tellus in conjunction with the components of Strode.

As we can see from the tables it was easier for Tellus to have an overview over these five components. This is probably because of the fact that they were so few people in relation to Omega. At Omega the components were easier fulfilled for each supplier than the overall project. This might have had something to do with the fact that they were focused on following the contracts. The last component, know who knows what, was difficult to answer for both projects. At Tellus there were only meetings observed, and

Table 5.2: Strode's Implicit Coordination Factors for Omega

Component	**Omega**
Know why	The project management was aware of the overall goal and how tasks contributed to the overall goal. The team members became aware eventually, first aware of the supplier and their goal, and eventually the whole project and the overall goal
Know what is going on and when	Each individual had the opportunity to se the burndownchart of other teams, and were aware of what was in their contracts, but it did not seem like they knew about all tasks in the project overall
Know what do to and when	They were to be aware of which tasks that should be prioritized, and they helped other teams when they had little to do, but they seemed to prioritize their team and supplier before the overall project
Know who is doing what	The team members were mostly aware of tasks the other team members or their supplier worked on. After a while they also knew if someone worked on something they had worked on previously
Know who knows what	This is not easy to see from the interviews, but they were aware of which supplier which had the responsibility of the different components

Table 5.3: Strode's Implicit Coordination Factors for Tellus

Component	**Tellus**
Know why	At Tellus it seemed like everyone was aware of the overall project goal and how tasks contributed to this through several discussions on dependencies
Know what is going on and when	Everyone were aware of all tasks in the backlog and were able to see the burndownchart and which squad were responsible for what
Know what do to and when	At one point Saturn did not have many tasks for the next sprint that were important to the project and therefore decided to help with Venus's tasks
Know who is doing what	Because there were few people it was easy for everyone to know who did what
Know who knows what	This was not possible to observe, but it seemed like they knew where to find the knowledge required

since this was not mentioned it is difficult to conclude something. At Omega there were so many people involved which makes it difficult, but they probably had an idea of who to ask because of the components they were responsible for.

5.2 Evaluation of the Study

Before concluding this study it is important to evaluate it. This is conducted by the researcher. The reason for doing this is to give an understanding about the research, especially the cases chosen. It is therefore important to build confidence and credibility in the study and the researcher. The research process will be discussed together with the principles of Klein and Myers before discussing if the cases are agile.

5.2.1 Research Process

This is a master thesis with a timeframe of 20 weeks. Because of these time constraints the researcher had to use material from the Omega case that had been conducted previously by other researchers. The researcher had to sign a non disclosure agreement, and was only given three focus group interviews in addition to background information in two other papers. When it came to Tellus, the researcher was able to observe four meetings, while another student had observed two additional meetings the researcher gained access to. The researcher could probably have observed additional meetings, but then there would have been less time to review the cases and comparing them. It would have been beneficial if there had been the possibility of interviews which could have given the researcher more information about the subject at hand.

Klein and Myers Principles

If we see this study together with Klein and Myers principles of interpretive field we can look more closely on the study conducted. The results can be seen in table 5.4.

5.2.2 Generalisation

The development at Omega was based on producing a new system with all functionalities of the old system with improvements because the old system was being phased out. It was therefore unique since everyone was located in not only the same building but the same room. There were several different suppliers in this project which had to work together. The fact that many were new to agile methodology is also an important aspect because of the rise in agile practitioners. At Tellus they were mostly creating a new system, with the reuse of code from other projects. The team is divided into two squads and they work distributed. There is only one supplier involved. For the sake of generalising the projects, the researcher believe that Tellus is the project which most easily can be generalised, but in both projects there should be possible to discover somewhat the same factors in other large scale agile projects.

Table 5.4: The Researcher's Use of the Principles of Klein and Myers

Principle	How the principles were used
The fundamental principle of the hermeneutic circle	The different cases were first seen together with the theory and then compared with each other. For each of the different success factors the cases and situations in them were described.
The principle of contextualization	Information on both cases and the reason for these projects is given by the researcher.
The principle of interaction between the researcher and subjects	The researchers understanding of the cases was developed through observations; Tellus, and interviews conducted by other researchers; Omega. A presentation was also held for Tellus where the subject agreed on the findings from the researcher.
The principle of abstraction and generalization	The findings are explained together with previous success factors presented in theory.
The principle of dialogical reasoning	The researcher have investigated whether or not the aforementioned success factors can be seen in the cases and how they have affected the cases, both in positive and negative.
The principle of multiple interpretations	At Tellus all research comes from observations and is therefore from the assumptions of the resarcher. At Omega the interviews have been completed after the project finished and with people from different teams that have had different roles during the project duration. The interviews were conducted by other researchers that have had other research than this in mind.
The principle of suspicion	At Tellus all views come from the researchers point of view, and other people could see things in a different manner. At Omega the interviews were with people on different teams and with different roles, but because of the size of the project it is not clear if the meanings expressed here are the meanings of the majority.

5.2.3 Are the Cases Agile?

According to Ambler [6] there are five criterias that decide if you are really agile. The first criteria considers whether the agile teams provide value to their customers on a regular basis. At Omega they were located together with regular meetings like demos. At Tellus they had regular backlog grooming meetings in addition to demos. Both projects therefore fulfill the value criteria.

The second criteria is that agile teams should do continous developer regression testing and that they should take a test driven development approach. This approach is not mentioned in either of the two cases, but we do know that at Omega they either have specialized testers or developers who also conducted the testing. The conclusion is therefore that Omega fulfills the validation criteria. At Tellus testing is not much mentioned and therefore it is unknown whether this criteria is fulfilled.

The third criteria involves the agile teams to work closely with their stakeholders, ideally on a daily basis. At Omega there were stakeholders present at all times and there was no problem to obtain them. At Tellus we know they had regular meetings, and from the observation we can read that they sometimes had contact with stakeholders through other means. Since we do not have the complete understanding for Tellus we are not one hundred percent sure if this is correct, but we know it is correct in some degree. For Omega we know that the criteria is fulfilled.

Self-organization is the fourth criteria. At Omega there was said that all teams were self-organizing and even though it was encouraged to pair program, each team was able to decide for themselves. At Tellus the team is divided into two squads. From the observations it would appear like the squads are self-organized with e.g. own sprint planning for squad. Therefore it seems as though this criteria is fulfilled for both, but since we are not certain when it comes to Tellus they are set to maybe.

The fifth and last criteria is that agile teams regularly reflect on, and measure how they work together and act to improve on their findings in a timely manner. At both projects they have retrospectives. At Omega we could see that not only those in the team ensured that the concerns were handled, this was also accomplished by others who read the retrospectives. At Tellus they divided the sprint into two parts when people had a problem with planning for four weeks at a time. Therefore we can say that both projects fulfill this criteria.

Table 5.5: Criterias to be Agile

Criteria	**Omega**	**Tellus**
Value	Yes	Yes
Validation	Yes	Unknown
Active stakeholder participation	Yes	Maybe
Self-organization	Yes	Maybe
Improvement	Yes	Yes

As we can see from table 5.5 we can confirm that Omega was definitely agile. We do not have all the facts about Tellus, but from what we have observed four out of the five criterias were fulfilled. Since these are only suggestions for a project being agile we can therefore conclude with the fact that both of the projects are agile.

Chapter 6

Conclusion

In this chapter we will try to conclude the findings from this thesis. We will do so in conjunction with the research question given earlier. In the end we will give an overview of implications for practice.

6.1 Research Question

> ***Is whole team thinking a possible success factor in large scale agile development?***

As we have witnessed there are several important success factors in large scale agile development. All of these do not have to be a success factor in every project, and some can be hindering even though the project is said to be a success. No two projects will ever be the same because of different people, motivations, experience etc. One thing it appears as they have in common is the necessity for whole team thinking.

From what we have discovered in the theory we have fourteen different possible success factors. When investigating these two cases we found eleven of these applicable in one or both of the cases. The reason for not finding the last three could have something with the focus, which were on whole team thinking. Out of the eleven factors there were nine more or less applicable in both cases, some of these were only mentioned once or twice in one of the cases and is therefore not presented in the results chapter. From what we have seen in these two cases and from the theory there are at least five of the factors that are factors in all projects, the rest are unsure, see table 6.1.

Whole team thinking does not only affect the factor itself but it can also be seen in conjunction with several of the other factors, see table 6.2. Whole team thinking is not visible, but seen between the lines. It is about everyone coming together and doing what is best for the project and not only the team. Without it there would be difficult not only to have a workable product, but to finish the project.

Table 6.1: Success Factors Applicable in all Large Scale Agile Projects: A = always applicable, M = maybe always applicable, NF = not found in the cases

Success factor	**Applicable**
Customer collaboration	M
Agile software engineering techniques and agile evangelist	A
Responding to change	M
Team	A
Leadership	M
Planning	M
Testing	M
Governance	NF
Divide after you conquer	NF
Continous integration	NF
Bridgehead	A
Co-location	M
Product owner	A
Whole team thinking	A

Table 6.2: Success Factors Whole Team Thinking have been seen in Conjunction with:
Y= yes, N= no, NK = not known

Success factor	**WTT**
Customer collaboration	N
Agile software engineering techniques and agile evangelist	N
Responding to change	Y
Team	Y
Leadership	Y
Planning	N
Testing	N
Governance	NK
Divide after you conquer	NK
Continous integration	NK
Bridgehead	Y
Co-location	Y
Product owner	Y

From what we have seen in both cases whole team thinking is a factor that affects the projects in one way or another. At Omega they saw the divison between the teams and worked toward removing these, while they at Tellus have tried to see themselves as one team from the beginning. Whole team thinking has definitely been a factor in both of these projects. This can be seen by the results from Strode's implicit coordination factors in table 5.2 and 5.3.

Tellus is not yet concluded and there is no way of knowing if it will be a success or not. This is not hindering when we conclude with whole team thinking as a success factor. As already determined there might not be necessary for all success factors in every project because of the different resources, duration, and technologies. There will still be necessary for whole team thinking even if it is seen in conjunction with the other factors. Whole team thinking is one of the factors that will remain in every project, and without considering it the project will probably fail because of different thoughts of direction between the teams.

From what we have seen both in the theory and in the cases is that it is hard to not have whole team thinking in mind when doing a large scale agile development project. Not all lessons learned-papers mention it, but it can usually be seen between the lines, and in conjunction with other factors for success. Therefore it is safe to say that whole team thinking is an important success factor in large scale agile development projects.

6.2 Implications for Practice

Now that we can see that whole team thinking is a possible success factor it is important to know how it can be accomplished. At the beginning of the project it is important to think about the shared mental models in the whole project and for each team. This should be conducted for both management, and for everyone involved in the project. This can ensure that all team members have an understanding of the project as a whole from the beginning.

It is important to find different arenas for the project to provide knowledge from every part of the team. This can be accomplished by for example scrum of scrums. The bigger the project, the more arenas are necessary for the whole project. The arenas work as a bridgehead between the teams and ensures coordination between them. The architects should talk together, and the same applies for testers, designers and others. It is very important that the arenas are not forced on the teams, and can change over time to best suit the project. There is no point in continuing with scrum of scrums if nobody has the need for it, it is therefore important to be able to respond to changes.

It is not necessary to only hire people with agile experience, as long as they are passionate about learning agile practices. When hiring people with agile experience it is important that they know that a large scale project is different from a project with only one team. The leaders should be able to unite the teams and ensure that they understand that they are all working on the same project, and how important everyone's contribution is.

The customer should dedicate their time to the project, because if they do not, the project will most likely fail. It is necessary that they use their best people with the right knowledge and authority to make decisions. The PO or POs all have to have an understanding that they are there for the whole project and not only for their expertise in the different areas of the project. They have to be able to collaborate and agree on which areas of the project to use the most resources.

Everyone should know what the other teams are working on and how this affects their work. In addition the team members should be encouraged to help if they have the capacity and knowledge. This can easily be seen through the shared mental models already from the beginning. The shared mental models provide the team members and teams with a common understanding of the task and the people and teams involved. Everything will be easier the more co-located everyone is, but as long as they have good replacements, like a video live-feed, it can work either way. Locating teams working on the same areas of the project together will help and can bring the teams closer together.

If we add up we can se that the most important practices for whole team thinking is:

- Shared mental models for the whole project
- Different arenas
- Listening to the teams and responding to changes
- Hire people with an understanding of the requirements necessary for a large project
- A customer who understands the needs in a large agile project

Here the shared mental models is one of the most important practices and is something that are not only necessary from the beginning, but throughout the whole project.

Chapter 7

Future Work

The future work chapter will propose further research areas and how to continue the research of whole team thinking in large scale agile development. First we will consider the revised research agenda from XP2014 before considering this thesis and its findings.

7.1 Suggestions for Future Research

Before proposing new suggestions we should look at the revised research agenda from Dingsøyr and Moe [2], see table 7.1. In this thesis we have researched the variability factors. Inter-team coordination have already been a topic widely researched, although there are different conclusions, for example if the scrum of scrums work [26; 56]. Knowledge sharing and improvement is something that can be seen together with the shared mental models and whole team thinking.

It is still difficult to find how to best scale agile practices, and comparing similar projects with different practices could be a way of studying this more closely. One of the topics actually mentioned by the Omega case is the contracts. People were concerned with delivering what was in the contracts which made it difficult to consider the project as a whole, and it made it difficult to rotate team members between suppliers.

From this thesis we can see that there are several of the success factors which are present in the two cases, and in the research agenda, and some that have not been found or discovered, see table 6.1 and 6.2. For future research it might be necessary to explore these both in a narrow and wide perspective.

When it comes to whole team thinking and shared mental models there are several interesting ways of discovering the necessity of these. One is to follow a project which is aware of the importance and which allocates time to nurture the idea. The outcome might be different when everyone involved is aware of the possible problem or factor from the beginning. If this is something that helps the project to more success would be an interest-

Table 7.1: Revised Research Agenda for Large Scale Agile by Dingsøyr and Moe

Priority	**Topic**	**Description**
High	Organisation of large development efforts	Organizational models, portfolio management, governance, project management, agile productline engineering
High	Variability factors in scaling	Factors important in large projects which influence development process
High	Inter-team coordination	Coordination between teams
High	Key performance indicators	Identify metrics to monitor progress and support transparency
High	Knowledge sharing and improvement	How to ensure feedback for learning, use of knowledge networks and learning practices
High	Release planning and architecture	Coordinating and prioritizing functional and non-functional requirements, continous delivery, minimizing technical debt
Medium	Customer collaboration	Practices and techniques for product owners and customers to collaborate with developers
Medium	Scaling agile practices	Which agile practices can scale and understand why and when they do
Medium	Agile contracts	Understand if contracts can change the mind-set of customers from upfront planning to agile principles and uncover legal limitations that reduce agility
Medium	Agile transformations	Efficient adoption of agile practices
Medium	UX design	Integration of user experience design

ing finding. Then it could be possible to measure the possible success rate in later projects.

This thesis followed two different cases. There were no interviews being conducted by the researcher. If the researcher had been able to do interviews that focused on the theme of this thesis, which are success factors and whole team thinking, the answers could have been something else. It might have been possible to see more of what they did to enable whole team thinking and it could have been less assumptions taken by the researcher. Since the one case was already concluded they could have contributed with suggestions to what could be conducted in future cases. With more focus on whole team thinking they might have more to contribute in later agile large scale projects.

It would also be interesting to see if the other factors not found in these cases are applicable in other projects. There might be different cases which have different factors important, and discovering what factors cause the different projects in their success. Some of the factors are so natural that they are not much considered during the project because they are obvious. With more focus on them it can help people in their next projects. In one project the fact that the product owner is involved may be taken for granted, which is why it is difficult to discover the lack of involvement in other projects.

As we can see there are still endless possibilities for further research in large scale agile development projects. Conducting the same research in more cases, explore the different factors and continuing to watch the agile development in large scale are just some possibilities. Every project will be different, not only because no one creates the same product, but because there are different people with different skills, and there is always new technologies. This is why there are so many possibilities for further research in the large scale agile development.

Bibliography

[1] T. Dingsøyr, S. Nerur, V. Balijepally, and N. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, 2012.

[2] T. Dingsøyr and N. B. Moe, "Towards principles of large-scale management: A summary of the workshop at XP2014 and a revised reasearch agenda," in *XP2014 Workshops*, 2014.

[3] K. H. Rolland, "Desperately seeking research on agile requirements in the context of large-scale agile projects," in *XP 2015 Workshops, May 25-29, Helsinki, Finland*, 2015.

[4] H. Kniberg and A. Ivarsson, "Scaling agile @ Spotify," 2012.

[5] J. Schnitter and O. Mackert, "Large-scale agile software development at SAP AG," in *5th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2011.

[6] S. W. Ambler, "IBM agility@scale: Become as agile as you can be," 2012.

[7] S. C. Misra, V. Kumar, and U. Kumar, "Identifying some important success factors in adopting agile software development practices," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1869–1890, 2009.

[8] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, vol. 81, no. 6, pp. 961–971, 2008.

[9] B. J. Oates, *Researching Information Systems and Computing*. SAGE Publications Ltd, 2006.

[10] T. C. Syversen, "Whole team thinking and possible success factors in large scale agile development projects." Preliminary paper at the Department of Computer and Information Science, NTNU, 2015.

[11] S. W. Ambler, “Agile software development at scale,” in *2nd IFIP Central and East European Conference on Software Engineering Techniques*, 2007.

[12] A. Shatil, O. Hazzan, and Y. Dubinsky, “Agility in a large-scale system engineering project: A case study of an advanced communication system project,” in *IEEE International Confererence on Software Science, Technology & Engineering*, 2010.

[13] S. Nerur and V. Balijepally, “Theoretical reflections on agile development methodologies,” *Communications of the ACM*, vol. 50, no. 3, pp. 79–83, 2007.

[14] J. Highsmith and A. Cockburn, “Agile software development: The business of innovation,” *Computer*, vol. 34, no. 9, pp. 120–122, 2001.

[15] J. A. Ingvaldsen and M. Rolfsen, “Autonomous work groups and the challenge of inter-group coordination,” *Human Relations*, vol. 65, no. 7, pp. 861–881, 2012.

[16] H. Saeeda, F. Arif, N. M. Minhas, and M. Humayun, “Agile scalability for large scale projects: Lessons learned,” *Journal of Software*, vol. 10, no. 7, pp. 893–903, 2015.

[17] C. T. Scmidt, T. Kude, J. Tripp, A. Heinzl, and K. Spohrer, “Team adaptability in agile information systems development,” in *34th International Conference on Information Systems, Milan*, 2013.

[18] C. T. Scmidt, S. G. Venkatesha, and J. Heymann, “Empirical insights into the perceived benefits of agile software engineering practices: A case study from SAP,” in *International Conference on Software Engineering*, 2014.

[19] D. Cohen, M. Lindvall, and P. Costa, “An introduction to agile methods,” *Advances in Computers*, vol. 62, pp. 1–66, 2004.

[20] L. Rising and N. S. Janoff, “The scrum software development process for small teams,” *IEEE Software*, vol. 17, no. 4, pp. 26–+, 2000.

[21] T. Dingsøyr and Y. Lindsjørn, “Team perfomance in agile development teams: Findings from 18 focus groups,” in *14th International Conference on Agile Processes in Software Engineering and Extreme Programming*, 2013.

[22] C. T. Scmidt, K. Spohrer, T. Kude, and A. Heinzl, “The impact of peer-based software reviews on team performance: The role of feedback and transactive memory systems,” in *33rd International Conference on Information Systems, Orlando*, 2012.

[23] T. Kude, S. Bick, C. Scmidt, and A. Heinzl, “Adaption patterns in agile information systems development teams,” in *22nd Conference on Information Systems, Tel Aviv*, 2014.

[24] E. Salas, D. E. Sims, and C. S. Burke, “Is there a big five in teamwork?,” *Small Group Research*, vol. 36, no. 5, pp. 555–599, 2005.

[25] T. Dybå and T. Dingsøyr, “Empirical studies of agile software development: A systematic review,” *Information and Software Technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[26] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large scale: A revelatory case study and research agenda for agile method adaption." Work in progress, 2016.

[27] A. Scheerer and S. Bick, "Five ways to scaling agile coordination in large-scale agile software development," 2015. SINTEF Symposium Nov 16, 2015 Trondheim powepoint presentation.

[28] H. Saeda and F. Arif, "Systematic literature review of agile scalability for large scale projects," *International Journal of Advances Computer Science and Applications*, vol. 6, no. 9, pp. 63–75, 2015.

[29] M. Giblin, P. Brennan, and C. Exton, "Introducing agile methods in a large software development team: The developers changing perspective," in *11th International Conference on Agile Software Development (XP2010)*, 2010.

[30] D. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE Software*, vol. 20, no. 4, pp. 12–14, 2003.

[31] M. Paasivara, V. T. Heikkilä, and C. Lassenius, "Experiences in scaling the product owner role in large-scale globally distributed scrum," in *IEEE Seventh International Conference on Global Software Engineering*, 2012.

[32] B. S. Blau, T. Hildenbrand, R. Knapper, A. Mazarakis, Y. Xu, and M. G. Fassunge, "Steering through incentives in large-scale lean software development," *Evaluation of Novel Approaches to Software Engineering*, vol. 275, pp. 32–48, 2013.

[33] A. Scheerer, T. Hildenbrand, and T. Kude, "Coordination in large-scale agile software development: A multiteam systems perspective," in *47th Annual Hawaii International Conference on System Sciences*, 2014.

[34] A. Scheerer and T. Kude, "Exploring coordination in large-scale agile software development: A multiteam systems perspective," in *35th International Conference on Information Systems, Auckland*, 2014.

[35] S. Bick, K. Spohrer, A. Scheerer, T. Kude, and A. Heinz, "Software development in multiteam systems: A longitudinal study on the effects of structural incongruences on coordination effectiveness," in *eProceedings of the 9th International Research Workshop on Information Technology Project Management Auckland, New Zealand, December 13th*, 2014.

[36] A. Scheerer, S. Bick, T. Hildenbrand, and A. Heinzl, "The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach," in *48th Hawaii International Conference on System Sciences*, 2015.

[37] D. E. Strode, S. L. Huff, B. Hope, and S. Link, "Coordination in co-located agile software development projects," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1222–1238, 2012.

[38] J. Vlietland and H. van Vliet, “Towards a governance framework for chain of scrum teams,” *Information and Software Technology*, vol. 57, pp. 52–65, 2015.

[39] J. Schnitter and O. Mackert, “Introducing agile software development at SAP AG: Change procedures and observations in a global software company,” in *5th International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, July 22-24*, 2010.

[40] C. Larman and B. Vodde, “Scaling agile development: Large and multisite product development with large-scale scrum,” *CrossTalk*, June 2013. Large Scale Agile.

[41] A. Martini, L. Pareto, and J. Bosch, “Improving business success by managing interactions among agile teams in large organizations,” in *4th International Conference on Software Business*, 2013.

[42] A. Koski and T. Mikkonen, “Rolling out a mission critical system in an agilish way,” in *IEEE/ACM 2nd International Workshop on Rapid Continous Software Engineering*, 2015.

[43] H. Koehnemann and M. Coats, “Experiences applying agile practices to large systems,” in *Agile Conference*, 2009.

[44] I. Gat, “How BMC is scaling agile development,” in *Agile Conference*, 2006.

[45] G. Benefield, “Rolling out agile in a large enterprise,” in *Proceedings og the 41st Hawaii International Conference on System Sciences*, 2008.

[46] M. R. J. Qureshi, “Agile software development methodology for medium and large projects,” *IET Software*, vol. 6, no. 4, pp. 358–363, 2012.

[47] M. Daneva, E. van der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar, N. Ajmeri, U. Ramteerthkar, and R. Wieringa, “Agile requirements prioritization in large-scale outsourced system projects: An empirical study,” *Journal of Systems and Software*, vol. 61, no. 5, pp. 1333–1353, 2013.

[48] A. W. Brown, “A case study in agile-at-scale delivery,” in *12th International Conference on XP*, 2011.

[49] E. Moore and J. Spens, “Scaling agile: Finding your agile tribe,” in *Agile Conference*, 2008.

[50] S. W. Ambler, “Supersize me.” http://www.drdobbs.com/supersize-me/184415491, 13.13 PM May 12th 2015.

[51] A. W. Brown, S. Ambler, and W. Royce, “Agility at scale: Economic governance, measure improvement, and disciplined delivery,” in *35th International Conference on Software Engineering*, 2013.

[52] E. Moore, “Influence of large-scale organization structures on leadership behaviors,” in *Agile Conference*, 2009.

[53] A. Elshamy and A. Elssamadisy, "Applying agile to large projects: New agile software development practices for large projects," in *8th International Conference on Agile Processes in Software Engineering and Extreme Programming*, 2007.

[54] M. Cataldo and J. D. Herbsleb, "Coordination breakdowns and their impacts on development productivity and software failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 343–360, 2013.

[55] K. Kaur, A. Jajoo, and Manisha, "Applying agile methodologies in industry projects: Benefits and challenges," in *International Conference on Computing Communication Control and Automation*, 2015.

[56] M. Paasivaara, C. Lassenius, and V. Heikkila, "Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work?," in *6th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2012.

[57] M. Paasivaara and C. Lassenius, "Communities of practice in a large distributed agile software development organization - Case Ericsson," *Information and Software Technology*, vol. 56, no. 12, pp. 1556–1577, 2014.

[58] M. Laanti, "Implementing program model with agile principles in a large software development organizations," in *Annual IEEE International Computer Software and Applications Conference*, 2008.

[59] J. Vlietland, R. van Solingen, and H. van Vliet, "Aligning codependent scrum teams to enable fast business value delivery, a governance framework and set of interventiuon actions," *Journal of Systems and Software*, vol. 113, pp. 418–429, 2016.

[60] T. E. Fægri and N. B. Moe, "Re-conceptualizing requirements engineering: Findings from a large-scale, agile project," in *XP 2015 Workshops, May 25-29, Helsinki, Finland*, 2015.

[61] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. Mary, and T. Kähkönen, "Agile software development in large organizations," *Computer*, vol. 37, no. 12, pp. 26–+, 2004.

[62] J. E. Hannay and H. C. Benestad, "Perceived productivity threats in large agile development projects," in *ESEM, September 16-17, Bolzano-Bozen Italy*, 2010.

[63] R. J. Stout, J. A. Cannon-Bowers, E. Salas, and D. M. Milanovich, "Planning, shared mental models, and coordinated performance: An empirical link is established," *Human Factors*, 1999.

[64] X. Yu and S. Petter, "Understanding agile software development practices using shared mental models theory," *Information and Software Technology*, vol. 56, no. 8, pp. 911–921, 2014.

[65] J. E. Mathieu, T. S. Heffner, G. F. Goodwin, E. Salas, and J. A. Cannon-Bowers, "The influence of shared mental models on team process and performance," *Journal of Applied Psychology*, vol. 85, no. 2, pp. 273–283, 2000.

[66] N. B. Moe, T. Dingsøyr, and T. Dybå, "A teamwork model for understanding an agile team: A case study of a scrum project," *Information and Software Technology*, vol. 52, no. 5, pp. 480–491, 2010.

[67] T. Dingsøyr and N. B. Moe, "Agile outside the comfortzone (smidig utenfor komfortsonen)," 2015. Smidig 2015, https://vimeo.com/album/3642046/video/144822306.

[68] P. Runeson and M. Host, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[69] T. Dingsøyr, T. Fægri, and J. Itkonen, "What is large in large-scale? a taxonomy of scale for agile software development," *Product-Focused Software Process Improvement*, vol. 8892, pp. 273–276, 2014.

[70] N. H. Wolfinger, *Qualitative Research*, ch. On writing fieldnotes: collection strategies and background expectancies, pp. 85–95. SAGE Publications, 2002.

[71] G. Guest, E. Namey, and M. Mitchell, *Collecting qualitative data: A field manual for applied research*, ch. Participant Observation, pp. 75–112. SAGE Publications, 2012.

[72] H. Klein and M. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systemst," *MIS Quarterly*, vol. 23, no. 1, pp. 67–93, 1999.

[73] E. Andreassen, "Inter-team coordination in large-scale agile software development - an exploratory case study," 2015. Master Thesis at the Department of Computer and Information Science, NTNU.

[74] D. Badampudi, S. A. Fricker, and A. M. Moreno, "Perspective on productivity and delays in large-scale agile projects," in *14th International Conference on Agile Processes in Software Engineering and Extreme Programming*, 2013.

Appendix A

Reasoning for Success Factors

In this appendix we will present the different articles where we found the success factors and the words or phrases which supports them. This chapter is to help the reader to know the reasons for the success factors presented in this thesis.

A.1 Overview of Articles for each Success Factor

Here we will summarize the different factors in different tables. We will present the words found in the paper that supports the factor, and which article it is from. We will also give an overview of the numder of referenced articles for each article and number of times the article have been cited by others. Several of the articles have a question mark where the number of times cited were to be mentioned. This is because no information has been found for them.

Table A.1: Customer Collaboration Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Customer collaboration	A survey study of critical success factors in agile software projects [8]	Journal	27 + 84
Customer collaboration	Agile Scalability for larger scale Projects: Lessons Learned [16]	Journal	64 + ?
Customer collaboration	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2
Customer collaboration	The Effects of Team Backlog Dependencies on Agile Multiteam Systems: A Graph Theoretical Approach [36]	Conference	46 + 0
Customer collaboration	Experiences in Scaling the Product Owner Role in Large-Scale Globally Distributed Scrum [31]	Conference	10 + ?
Customer collaboration	Rolling out a mission critical system in an agilish way [42]	Conference	4 + ?
Part-time experts	Improving Business Success by Managing Interactions among Agile Teams in Large Organizations [41]	Conference	26 + 1

Table A.2: Agile Software Engineering and Agile Evangelist Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Agile software engineering techniques	A survey study of critical success factors in agile software projects [8]	Journal	27 + 84
Agile software engineering techniques	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Deliveryt [51]	Conference	32 + 2
Agile evangelist	How BMC is Scaling Agile Development [44]	Conference	0 + 2
Agile coaching	Rolling out Agile in a Large Enterprise [45]	Conference	5 + ?

Table A.3: Responding to Change Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Embrace change	Agile requirements prioritization in large-scale outsourced system projects: An empirical study [47]	Journal	30 + 13
Responding to change	Agile software development methodology for medium and large projects [46]	Journal	30 + 1
Responding to change	Agile Scalability for larger scale Projects: Lessons Learned [16]	Journal	64 + ?
Responding to change	The Effects of Team Backlog Dependencies on Agile Multiteam Systems: A Graph Theoretical Approach [36]	Conference	46 + 0
Changes	Exploring Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective [34]	Conference	67 + ?

Table A.4: The Team Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Team capability	A survey study of critical success factors in agile software projects [8]	Journal	27 + 84
Self-organizing scenario-based teams	Aligning codependent Scrum teams to enable fast business value delivery: a governance framework and set of intervention actions [59]	Journal	71 + ?
Self-organizing scenario-based teams	Agile Scalability for larger scale Projects: Lessons Learned [16]	Journal	64 + ?
Self-organizing scenario-based teams	Agile Software Development at Scale [11]	Conference	17 + 3
Self-organizing scenario-based teams	A Case Study in Agile-at-Scale Delivery [48]	Conference	8 + 3
Self-organizing scenario-based teams	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
Self-organizing scenario-based teams	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2
Self-organizing scenario-based teams	IBM agility@scale: Become as Agile as You Can Be [6]	Lessons learned	10 + ?
Right people	Scaling Agile: Finding your agile tribe [49]	Conference	3 + 4
Avoid ”super team”	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2
Good people	Supersize Me [50]	Web	6 + ?

Table A.5: Leadership Articles

Word/phrase	Article	Type of article	References + # of times referenced
Project manager	Applying Agile to Large Projects: New Agile Software Development Practices for Large Projects [53]	Conference	13 + 2
Good leader	Influence of Large-Scale Organization Structures on Leadership Behaviors [52]	Conference	10 + 1

Table A.6: Planning Articles

Word/phrase	Article	Type of article	References + # of times referenced
Project planning	Agile software development methodology for medium and large projects [46]	Journal	30 + 1
Planning	Perspective on Productivity and Delays in Large-Scale Agile Projects [74]	Conference	34 + 1
Two-level planning	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
Adaptive planning	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2

Table A.7: Testing Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Continous testing	Agile Scalability for larger scale Projects: Lessons Learned [16]	Journal	64 + ?
Test driven development	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
Test driven development	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2
Test driven development	IBM agility@scale: Become as Agile as You Can Be [6]	Lessons learned	10 + ?
Test cases	Empirical Insights into the Perceived Benefits of Agile Software Engineering Practices: A Case Study from SAP [18]	Conference	17 + 0

Table A.8: Governance Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Governance framework	Towards a governance framework for chain of Scrum teams [38]	Journal	99 + 2
Governance	Aligning codependent Scrum teams to enable fast business value delivery: a governance framework and set of intervention actions [59]	Journal	71 + ?
Economic governance	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
IT governance	Agile Software Development at Scale [11]	Conference	17 + 3
Effective governance	IBM agility@scale: Become as Agile as You Can Be [6]	Lessons learned	10 + ?

Table A.9: Divide after You Conquer Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Divide after you conquer	Applying Agile to Large Projects: New Agile Software Development Practices for Large Projects [53]	Conference	13 + 2
Divide and conquer	Supersize Me [50]	Web	6 + ?

Table A.10: Continous Integration Articles

Word/phrase	Article	Type of article	References + # of times referenced
Continous integration	Towards a governance framework for chain of Scrum teams [38]	Journal	99 + 2
Continous integration	Agile Scalability for larger scale Projects: Lessons Learned [16]	Journal	64 + ?
Continous integration	Scaling Agile: Finding your agile tribe [49]	Conference	3 + 4
Continous integration	Agile Software Development at Scale [11]	Conference	17 + 3
Continous integration	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
Continous integration	Experiences Applying Agile Practices to Large Systems [43]	Conference	3 + 2
Continous integration	Large Scale Agile: Scaling Agile Development: Large and Multisite Product Development with Large-Scale Scrum [40]	Lessons learned	6 + ?

Table A.11: Bridgehead Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Communities of practice	Communities of practice in a large distributed agile software development organization - Case Ericsson [57]	Journal	33 + 1
Scrum of scrums	Towards a governance framework for chain of Scrum teams [38]	Journal	99 + 2
Scrum of scrums	Exploring Software development at the Very Large Scale: A Revelatory Case Study and Research Agenda for Agile Method Adaption [26]	Journal	56 + ?
Scrum of scrums	Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work? [56]	Conference	14 + 4
Scrum of scrums	Large-Scale Agile Software Development at SAP AG [5]	Conference	20 + 3
Scrum of scrums	Exploring Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective[34]	Conference	67 + ?
Scrum of scrums	The Effects of Team Backlog Dependencies on Agile Multiteam Systems: A Graph Theoretical Approach [36]	Conference	46 + ?
Scrum of scrums	Applying Agile Methodologies in Industry Projects: Benefits and Challenges [55]	Conference	9 + ?
Bridgehead	Improving Business Success by Managing Interactions among Agile Teams in Large Organizations [41]	Conference	26 + 1
Cross-team activities	Scaling Agile: Finding your agile tribe [49]	Conference	3 + 4
Guilds, chapters	Scaling Agile @ Spotify [4]	Lessons learned	0 + 2

Table A.12: Co-Location Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Co-locating	Large-Scale Agile Software Development at SAP AG [5]	Conference	20 + 3
Co-location	Improving Business Success by Managing Interactions among Agile Teams in Large Organizations [41]	Conference	26 + 1
Geographical distribution	Agile Software Development at Scale [11]	Conference	17 + 3
Locating teams together	Scaling Agile @ Spotify [4]	Lessons learned	0 + 2
Scattered members	Scaling Agile Development: Large and Multisite Product Development with Large-Scale Scrum [40]	Lessons learned	6 + ?

Table A.13: Product Owner Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Product owner	Aligning codependent Scrum teams to enable fast business value delivery: a governance framework and set of intervention actions [59]	Journal	71 + ?
Product owner	Large-Scale Agile Software Development at SAP AG [5]	Conference	20 + 3
Product owner	Applying Agile to Large Projects: New Agile Software Development Practices for Large Projects [53]	Conference	13 + 2
Product owner	A Case Study in Agile-at-Scale Delivery [48]	Conference	8 + 1
Product owner	Implementing Program Model with Agile Principles in a Large Software Development Organizations [58]	Conference	19 + ?
Product owner	Experiences in Scaling the Product Owner Role in Large-Scale Globally Distributed Scrum [31]	Conference	10 + ?
Product owner	Rolling out a mission critical system in an agilish way [42]	Conference	4 + ?
Product owner	Scaling Agile @ Spotify [4]	Lessons Learned	0 + 2
Product owner	Large Scale Agile: Scaling Agile Development: Large and Multisite Product Development with Large-Scale Scrum [40]	Lessons learned	6 + ?

Table A.14: Whole Team Thinking Articles

Word/phrase	**Article**	**Type of article**	**References + # of times referenced**
Project overall	Agile Software development in Large Organizations [61]	Journal	9 + 45
Shared mental models	Aligning codependent Scrum teams to enable fast business value delivery: a governance framework and set of intervention actions [59]	Journal	71 + ?
Whole team thinking	Scaling Agile: Finding your agile tribe [49]	Conference	3 + 4
Project as a whole	Large-Scale Agile Software Development at SAP AG [5]	Conference	20 + 3
Full lifetime cycle	Agile Software Development at Scale [11]	Conference	17 + 3
Collaborative teamwork	Agility at Scale: Economic Governance, Measure Improvement, and Disciplined Delivery [51]	Conference	32 + 2
Overall situation	Improving Business Success by Managing Interactions among Agile Teams in Large Organizations [41]	Conference	26 + 1
Whole product	Large Scale Agile: Scaling Agile Development: Large and Multisite Product Development with Large-Scale Scrum [40]	Lessons learned	6 + ?
Share philosophical mindset	Supersize Me [50]	Web	6 + ?

Appendix B

Observation Template

In this chapter the reader can see the observation template used during the observations at Tellus. The observations were conducted in Norwegian, the native language of the observer and the subjects. The first page consists of various information which can be important for the observer at a later time. The next page is the observations conducted. Here we have an overview of what was said and done during what time. When observing there were several copies of the second page included.

B.1 The Template

Observasjon av:

Dato og tid:

Sted (antall):

Bedrift:
Case:
Personer:

Sitteplasser:

Vindu:
Vær:
Bakgrunnsstøy:
Humør/form:
Romtemp:
Teknisk utstyr:

Div:

Kl:	Observasjon:

p

p

Appendix C

Supporting Information

This chapter includes some theory which were originally in the thesis. The reasons for removing this from the thesis was because it was not important for the research question. The reasons for having it in the appendix is because it supports some of the items that are presented in the thesis.

C.1 Agile Software Development

Here we will present both XP and enterprise agile. XP will be presented because it is mentioned in section 2.4.2. For those readers that do not know about XP when reading this thesis an explanation of it is given here. Enterprise agile is briefly mentioned in the thesis in section 2.2. This is therefore explained here so that it is easy to understand the difference between enterprise agile and large scale agile.

C.1.1 XP

Extreme programming is often referred to as XP, see figure C.1[1] for an overview. XP consists of practices that focus on software development team activities [64]. It is close to being a full-fledged disciplined agile delivery method, but is missing explicit project initiation and release practices [6]. XP has the shortest recommended iteration length of the agile methods with only two weeks [19]. The original XP recipe is based on four simple values and twelve supporting practices[2]:

- The planning game
- Small releases
- System metaphor

[1]https://blogs.msdn.microsoft.com/jmeier/2014/06/06/extreme-programming-xp-at-a-glance-visual/
[2]https://www.versionone.com/agile-101/agile-methodologies/

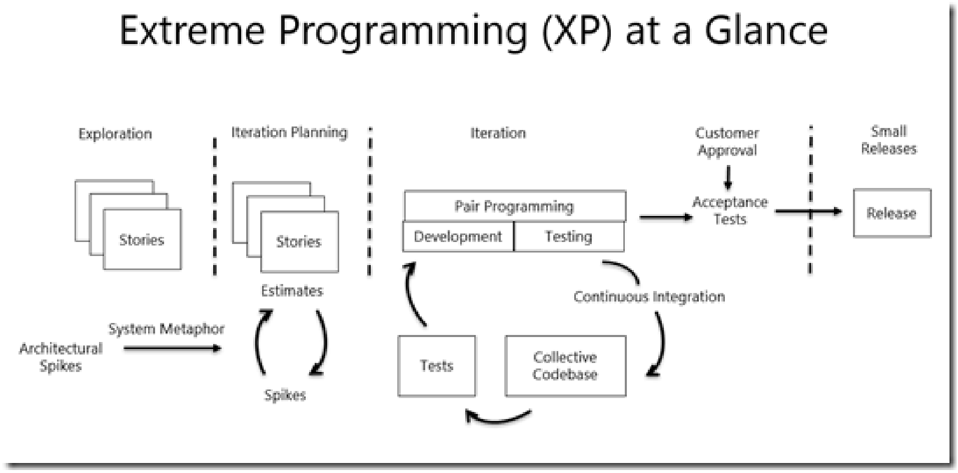

Figure C.1: Extreme Programming Overview

- Simple design
- Tests
- Refactoring
- Pair programming
- Continous integration
- Collective code ownership
- On-site customer
- 40-hour weeks
- Open workspace

The four values are; simplicity, communication, feedback and courage.

The basic advantage of XP is that the whole process is visible and accountable[3]. In XP there are two planning steps. Release planning is where the customer presents the desired features to the programmers. Here the programmers estimate the difficulty of these features. It is important to know that the release plan is revised regularly. The second step is iteration planning where the customer presents the features desired for the next iteration. In XP a feature story is never ninety percent complete, it is either finished or not[4].

XP is noted for pair programming and it advocates pair programming for feedback [14]. Instead of using a product backlog, XP uses story cards, but the size of the team is

[3]http://c2.com/cgi/wiki?ExtremeProgramming
[4]http://ronjeffries.com/xprog/what-is-extreme-programming/

about the same as in scrum with maximum ten people. XP has a focus on communication and co-location. This indicates that XP does not support distributed teams [19].

C.1.2 Enterprise Agile

Since agile software development has become widely popular several organizations have wanted to take part in the agile revolution. Dingsøyr et al. [26] refer to enterprise agile as agile methods applied in large organizations. How this can be accomplished is explained with Yahoo as an example [45]. The enterprise usually consists of several small projects which all use agile development methods. This means that the enterprise can have several independent projects which all use agile methods, but they are not connected. If the organization had had only one big project this would have been large scale and not enterprise agile. Large scale is explained in section 2.2.

A number of organizations have taken an interest in agile methods because of the necessity to seek alternatives to the traditional software development methodologies [61]. The traditional ones are often too cumbersome, bureaucratic and inflexible, and the organizations feel pressure to produce more at lower cost. This can be seen by this quote from a team at Motorola:

> *"Software development teams face a continous battle to increase productivity while maintaining or improving quality"*

The problems in organizations is that a project cannot truly be independent, but it must interact and follow the rules of the organization overall [61].

Agile practices have been implemented not only at Yahoo, but also in other organizations, or parts of organizations. Lindvall et al. [61] followed ABB, Daimler-Chrysler, Motorola and Nokia in their pilot projects. BMC [44] and IBM [48] are among the other companies that have implemented agile practices and are satisfied with the results.

t

Appendix D

Success Factors not Discussed in the Thesis

In this appendix we will give an overview of the success factors that had little or no data found in the two cases by the researcher. This does not imply that they are not success factors, or that there is not possible to find them in the cases.

D.1 Success Factors

First we will give an overview of the two factors with results from the cases, these are planning and testing. Both of this are applicable for both small and large scale and is followed by governance which is also applicable by both. The two last factors are most applicable for large scale and they are: divide after you conquer, and continous integration.

D.1.1 Planning

For this factor there are some results from the two cases. Since the results were not relevant for whole team thinking it was removed from the thesis and put in the appendix.

Theory

Like with many of the other factors planning is important, and it is particularly important in large projects since there are so many factors to consider. Project planning plays an important role for the success or failure of a project [46]. The objective in this phase is to concentrate on the major milestones of the project. If the project planning is not properly made and documented, a software project might deterioate. Large systems requires a planning process that accomodates changes and adapts to those [43]. All key stakeholders should be involved in the planning [74]. In large projects the planning will take place on different levels in the hierarchy. Top-down planning is an example of this since it is conceptualized as a mechanistic, centralized approach, e.g. between a team and a superior person or team [34]. Brown et al. [51] found five core practices that constitute the key prerequisites for efficient adoption of agile approaches, and two-level planning was one of these.

Results

The planning at Omega was implemented differently at the three suppliers. At Beta they only used a couple of hours planning the sprint, while at Alpha they used about a day. Alpha used a lot of the time to detail and ensure that everyone had the same understanding of the tasks. This made it possible for everyone to work on all tasks since they all had decided how to do them. Alpha felt better prepared, while Beta did not find it necessary to use that much time on planning. They believed more in doing the work and plan if problems emerged.

The meeting agenda and meeting structure seemed to be lacking at Tellus. The meetings were completed differently each time and there were often silent breaks when people did not know what to do next. They were not tough enough when it came to how much time they should use on the different items, which sometimes led them to not finish in time. This was especially a problem when they were to have another meeting next. In addition it was hard for people getting the answers to their questions, it sometimes appeared that no one cared about answering the question or knew how to answer. Other problems at Tellus included the fact that it was hard to plan a whole month ahead in a sprint, which eventually led to the sprint being divided into two parts.

D.1.2 Testing

For this factor there are some results from the two cases. Since the results were not relevant for whole team thinking it was removed from the thesis and put in the appendix.

Theory

One of the nine challenges found by Saeeda et al. [16] is challenges in regard to continous testing. In Ambler's [6] five criterias for being agile, validation is one of them and focuses on continous developer regression testing and that the agile teams should take a test driven development (TDD) approach. Brown et al. [51] also propose test driven development as one of the five core practices that constitute the key prerequisites for efficient adoption of agile approaches. Giblin et al. [29] interviewed developers after they had had a training course on test driven development. They felt that it could improve quality, but what concerned them was the lack of test harnesses for the legacy code.

Test driven development can help validate requirements/ scenarios early in the development process [43]. It is a technique where you write a single test and just enough production code to fulfill that test [11]. Test cases help developers mitigate errors and consequently minimize rework [18]. Test driven development helps agile teams capture detailed specifications in the form of executable tests instead of static documents or models [11]. In addition Brown et al. [51] present the importance of doing integration testing before unit testing. The scope of a unit is not necessarily a class. On large systems, units are typically a system component with significant behaviours or an interface to other parts of the system [43].

In a case study from SAP [18] they saw that high test coverage helped the team to deliver better software quality. It therefore helped to reduce the work stress during development sprints and before the end of the releases. This is just one of the cases that displays how important it is to test from day one since we are continously integrating [43]. This is a fact that is important no matter the size of the project.

Results

When it came to testing there were different practices at Omega. At Alpha they had specialized testers, while at Beta they rotated the role.

> *"To become a specialist you first have to be a generalist"*
> Project management at Alpha

It was important that the tester understands some code and have somewhat of a technical background like running SQL and deploy. It was important at Alpha that even though they had specialized testers that it was still up to the team to deliver, not any individual. The tester cannot do all tests in a few days. It was important that the tester was able to start as soon as a task was finished, if not they would not have had anything to do for the first half of the sprint.

D.1.3 Governance

If you have one or more IT projects then you have an IT governance in place [11]. Governance is critical to the success of any IT departments, and especially important at scale. Agile approaches to governance are based on collaborative approaches which enable teams to do the right thing. Governance can be classified as:

> *"Systematically determining who makes each type of decision (a decision right), who has input to a decision (an input right) and how these people (or groups) are held accountable for their role"* [59].

Ambler proposes [6] that to succeed at scaling agile you will require tools that integrate easily, and are sufficiently instrumented to provide the metrics required for effective governance.

According to Brown et al. [51] it is important to steer using economic governance, measure incremental improvements honestly, and empower teams with disciplined agile delivery. The reason for steering with economic governance is that it is an objective economic foundation for planning, decison making, and progress reporting that resolves uncertainties earlier and unifies constituencies on a shared set of expected target outcomes. From past measurement approaches they suggest three genres to software governance:

- Engineering governance

- Hybrid governance
- Economic governance

Effective governance is not about command and control, but it focuses on enabling the right behaviours and practices through collaborative and supportive techniques [11] which is why it is important both in small and large scale.

D.1.4 Divide after You Conquer

There is no such thing as a 200-person project team, one big project results in many sub-projects. This indicates that when you scale it is important to divide and conquer [50]. Elshamy and Elssamadisy [53] recommend to start with a core team that builds out a valid simple business case in a test driven manner. This also involves building out most of the architecture. This first phase ends when there is a stable code base with a significant portion of the architecture built out. After this first phase is finished the problem is said to be conquered and it is time to divide by growing the development team and splitting up into smaller subteams. Each subteam follows an agile process and interface with the other teams. There is challenges with having a master build and sustaining it, but this is one way of starting a large scale project which might prove successfull.

D.1.5 Continous Integration

One of the two key project challenges Moore and Spens [49] met in a large scale global development effort was continous integration. One of the seven areas SAFe targets is continous integration [16; 38]. Continous integration is presented as one of the five core practices that constitute the key prerequisites for efficient adoption of agile approaches by Brown et al. [51]. Continous integration is a development practice where developers integrate their work frequently, at least daily, where the integration is verified by an automated build [11]. In small scale this is not a problem since there are so few teams, but in large scale this is complicated since all code from the different teams are supposed to work together. Continous integration is important for the coordination of the teams and is often verified with automated tests, with a ”stop and fix” culture of rapidly fixing a broken build [40].

Large systems have complex integration challenges such as the shear volume of code and the variety of code sources. Because integration problems are exacerbated the longer integration is delayed, most large systems must perform some form of continous and early integration to reveal integration problems [43]. Moore and Spens [49] describe in their paper that the teams failed to monitor the build beyond their own module-level build. After a while there was a passionate core of individuals that were made into an integration team. They were responsible for ensuring the integrated builds were successfull. They were hesitant in creating this group because they hoped that team ownership of code would drive people to solve integration challenges. Sadly it was difficult to expand that passion and sense of responsibilty across the entire project team of over 300 people.

Koehnemann and Coats [43] proposed the following ideas between the developers and development teams to help facilitate continous and early integration:

- Organize teams around building scenarios, not components
- Solicit stakeholder feedback as often as possible
- Do not call everything a release
- Simplify the process of making changes
- Get all team members on board with continous integration

The reasons for building scenarios and not components is the fact that component teams can become self-focused and more concerned about their part of the system than the system's overall success [43]. Stakeholder feedback early in the integration process can help prevent future increments from building incorrect or undesirable implementations. When it comes to the last item we can see this together with the experiences from Moore and Spens [49]. Some individuals naturally resist, perhaps unintentionally, the change towards continous integration [43]. As we can see continous integration is essential to large scale since the time of integrating increases heavily with the time since the last integration.

Z