NTNU – Trondheim
Norwegian University of
Science and Technology

Coordination Between Teams in Agile Projects

Nikolai Hyldmo

Master of Science in Engineering and ICT
Submission date: June 2015
Supervisor: Nils Olsson, IPK
Co-supervisor: Torgeir Dingsøyr, Sintef IKT

Norwegian University of Science and Technology
Department of Production and Quality Engineering

TPK4920 – Project and Quality Management, Master's Thesis

Master thesis:

Coordination Between Teams in Agile Projects

Nikolai Hyldmo
Engineering and ICT, Production and project management
(Ingeniørvitenskap og IKT, Produkt og prosess)

NTNU
Department of Production and Quality Engineering

Spring 2015

Preface

This master thesis is submitted as the final work for the 5 year civil engineer study program "Engineering and ICT" at Norwegian University of Science and Technology (NTNU – Norges Teknisk-Naturvitenskapelige Universitet).

The thesis is written mainly as a literature study, but I have also looked at interviews done by Torgeir Dingsøyr with Norwegian companies that are performing large IT-projects. This has given me a unique insight in how project management and team coordination actually is done. This insight is a crucial part of understanding the actual dynamic and the issues that rises in the real world.

The work on this thesis had a quite slow start mainly because I chose to not continue on the project thesis from the previous semester, and therefore had to start from scratch in January to find a supervisor and a suitable topic. Furthermore, I started out writing a problem statement, and a pre-study report. Additionally I have spent a lot of time conducting literature search, and read about the topics, as I does not have lot of experience participating in agile projects. This could be considered a minor drawback in order to fully understand the topics I have been reading about. Especially in the first month, I did not feel entirely qualified to write about the topic. However, after spending a majority of the time reading about agile methods, team-coordination and peripheral topics, I acquired a better understanding of the domain, and were able to start writing.

I would like to thank the master thesis supervisor, Nils Olsson, for assistance and guidance regarding the thesis, and for valuable and quick feedback during the final stages. Additionally I would like thank Torgeir Dingsøyr for his advice and the interview material he gave me access to.

Summary

The way many software projects are managed have changed dramatically due to the introduction of agile methods. This shift was influences by the fact that technology and the IT-industry itself are moving rapidly and projects have to respond to rapidly changing requirements (Lindvall et al., 2002). Agile methods were a reaction to the plan based approaches that is not welcoming change in the same way as agile methods. According to Cockburn and Williams (2003) agile "is about feedback and change".

Many articles are referring to something called the "agile sweet spot" (Nord et al., 2014) (Baskerville et al., 2010). This is when the project only consist of up to 20 people and it is not a critical project in terms of serious money or lives at risk. However, many projects are larger and thus needs more than 20 persons to succeed, and it is still desirable to apply agile methodologies. This introduces new issues for project managers on how to effectively coordinate large projects consisting of many teams working in parallel.

Coordinating large, multi-team projects is not a new topic, but doing it in *agile* projects is a new issue. Agile projects follow a set of principles defined in the Agile Manifesto (Beck et al., 2001), and these are to be followed when coordinating teams in agile projects. Due to the fact that agile methods amongst other rely on "individuals and interactions over processes and tools" (Beck et al., 2001), it is not considered appropriate to enforce any specific working method onto the project teams. The teams are supposed to be self-organizing and autonomous (Nerur et al., 2010).

This master thesis presents agile methods in general and principles of coordination (both traditionally and in agile contexts). By studying literature and reading interviews with organizations, the goal was to find out how team coordination is done, or may be done in an effective and efficient manner, by using agile methods.

By definition all projects are unique, thus there are no general way of coordination a large project. However, there frameworks and methodologies that makes the coordination and agile implementation process easier. Software projects are well known for having lower success rate and more frequently budget overruns, compared to projects ran in other industries. The reason for this may be the fact that the technology industry is fast moving,

and the competition is tough. In order to deal with this, it is vital to be able to perform project management the best possible way, and be able to adapt to, and respond to change.

Keywords: Agile Software Development, Large-Scale Agile, Scaled Agile, Coordination of Teams, Communities of Practice, Software Architecture.

Table of Contents

Preface II
Summary III
List of figures VII
1 Introduction 1
1.1 Background 1
1.2 Problem statement 3
2 Methodology 6
3 Theory 7
3.1 The introduction of agile methods 7
3.2 Predecessors of agile methods 9
3.3 The introduction of agile 10
3.4 Different agile software development methods 12
3.4.1 Scrum 12
3.4.2 Scrum of scrums 17
3.4.3 Crystal methods 18
3.5 Traditional coordination of teams 19
3.5.1 Communities of practice 19
3.6 Coordination of teams in agile projects 22
3.6.1 Software tool support for agile organizations 22
3.6.2 Large-Scale scrum 24
3.6.3 Kanban and Scrum-ban 28
3.6.4 Software systems for coordinating 29
3.6.5 Software architecture 29
4 Results 32
4.1 Case study 32
4.1.1 Organization “Alpha” 32
4.1.2 Organization “Beta” 35
5 Discussion 37
6 Conclusion 41
6.1 RQ1: How is coordination between agile teams performed? 41
6.2 RQ2: What are the alternatives for coordination between agile teams? 41

6.3 RQ3: Are the coordination methods unique for agile projects, or are they well known from other industries as well? 42
6.4 RQ4: Is there an approximate linear correlation between number of teams (or number of team members), and how strict the coordination are? 43
7 References 44
8 Appendices 48
8.1 Appendix 1: Interview guide 48

List of figures

Figure 1 – Example of traditional project structure 4
Figure 2 – Example of two-team setting 4
Figure 3 – Illustration of more complex project organization 5
Figure 4 – Illustration of Framework-1 (Larman and Vodde, 2013) 25
Figure 5 – Illustration of Framework-2 (Larman and Vodde, 2013) 27
Figure 6 – Illustration of the different cases where the various team structures suit best (Eckstein, 2014) 30
Figure 7 – Illustration of different ways of assigning tasks (Bick et al., 2014) 39

1 Introduction

1.1 Background

Agile methods are quite well established in projects with only one development team. A term called the “agile sweet spot” (Baskerville et al., 2010) is often being referred to in articles, and this is when a team consists of about 20 people, and when the system being developed is not life-critical. However agile methods are not that well established in situations where there are large and multiple teams within the same project.

The Agile Manifesto lists four main points (Beck et al., 2001), stating what they prefer in an agile setting (for instance “***responding to change*** over following a plan”). In addition, there are 12 principles for agile software development. These are general guidelines that do not mention specific methods for how to pursue these. The principles can easily fit an organization, not only a simple team.

When you look into more specific agile frameworks like Scrum or XP, they are more focused on processes a team uses internally. Examples are pair-programming in XP and standup meetings within the team in Scrum. These are procedures that only apply to in a given scale. Which means you cannot easily scale the concept of pair-programming to the entire organization.

If you work on a project with a lot of teams, you cannot have one single standup meeting for all the teams and the team members, because it will be too time consuming, and it will easily exceed the 15 minutes of allocated time. This will work against it purpose of having standup meetings as a quick way to update and get updated on what the co-workers are doing. The goal of scrum is to quick and easily exchange information that the other project members would benefit from, instead of traditional, long meetings where some people tends to end up not doing anything productive.

Over the last 5-10 years, methodologies like “Scaled agile” and similar, have got more focus. These methods are concerned about using agile methods in larger contexts then singly projects, for instance over the entire organization, or over project portfolios. This report is supposed to assess how agile methods can be applied and implemented on larger projects

that consist of multiple teams. Additionally it will look into how these teams should coordinate and communicate amongst each other by taking advantage of agile methods.

According to (Dingsøyr and Moe, 2014), “organization of large development efforts” and “inter-team coordination” are among the topics that are rated with high priority on their list of topics that should be prioritized in the future. One of the reasons for this is the fact that the amount of academic studies on large scale agile is still very scarce (Paasivaara and Lassenius, 2014). However there are some practitioner literature advising on how to scale agile to larger context, but they are mostly written by consultants (Paasivaara and Lassenius, 2014).

1.2 Problem statement

The problem statement is “Coordination between teams in agile projects”. The goal of this thesis is to take a closer look at how people work in projects consisting of multiple teams. Agile methods are a group of many frameworks or types of approaches to agile methods. These frameworks and approaches all have their focus areas, where some focus on the individual and some focus on the team and so on. However, none of them seems to address how projects consisting of multiple teams should work together and how they communicate.

The thesis aims to find out what principles exists when it comes to coordination of teams, how team coordination is performed in projects. It will also be desirable if the methods used to coordinate team can be considered “agile” or if they are more traditional. Based on this some questions were established:

- **RQ1:** How is coordination between agile teams performed?
- **RQ2:** What are the alternatives for coordination between agile teams?
- **RQ3:** Are the coordination methods unique for agile projects, or are they well known from other industries as well?
- **RQ4:** Is there an approximate linear correlation between number of teams (or number of team members), and how strict the coordination are?

It will be relevant to take a look at companies that have and are currently performing larger IT-projects and get statements from stakeholders in the projects.

By definition, all projects are unique by their nature, and it is therefore probable that some projects are would be easy to coordinate even though it consist of many teams, while other projects is hard to coordinate. It may depend on how much the tasks of the different teams depends on each other.

If team A and Team B works on task A and task B respectively, and these tasks does not have a lot to do with each other, they will be easier to coordinate, compared to a situation where task A and task B are closely related. It will be interesting to talk to an organization who run these types of projects, and ask them how they do it.

The scope of the thesis is mainly IT-projects, but it could be relevant to look at projects in other industries if it suits the thesis.

Figure 1 – Example of traditional project structure

Figure 1 is showing the most basic structure of a project, consisting of a project team and a project manager. The project manager is usually the one responsible for communicating with the project owner and the stakeholders.

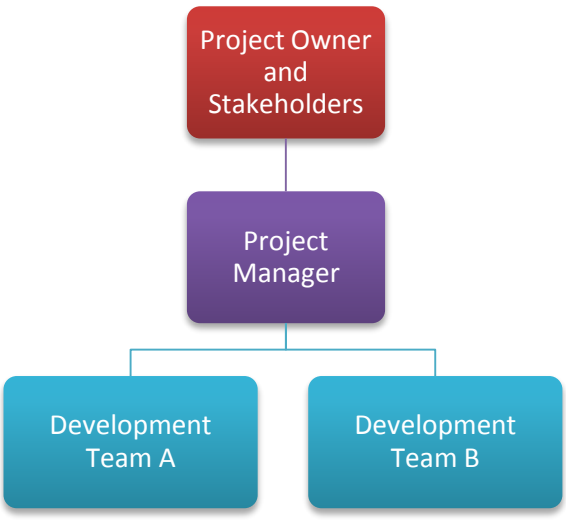

Figure 2 – Example of two-team setting

Figure 2 is showing a setting where there are two teams. The roles are slightly more undefined when it comes to communication and coordination of the team. For instance if communication should go through the project manager, or if the teams should communicate directly with each other instead.

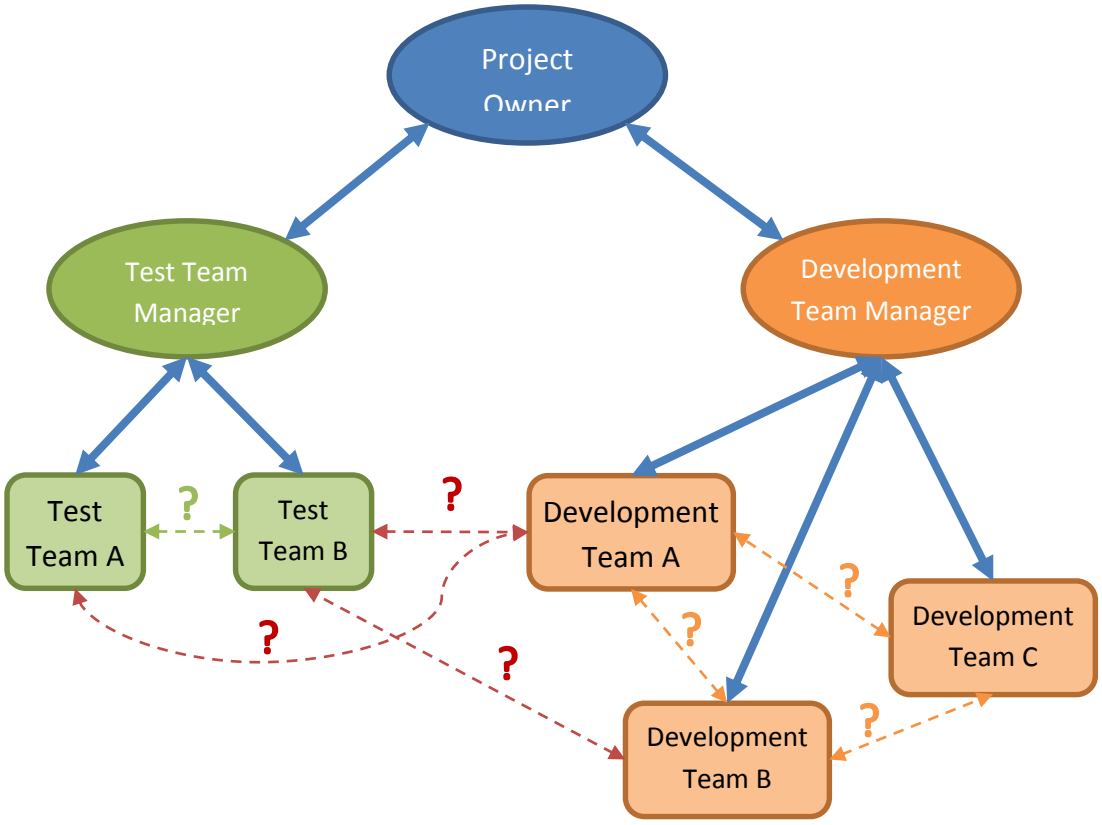

Figure 3 – Illustration of more complex project organization

Figure 3 shows a more complex project organization, with possible communication lines between teams, as well as through the regular manager. One of the goals of the thesis is to find out if this is how communication between teams are performed.

2 Methodology

The methodology of this thesis is mainly a literature study, and it will refer to past studies and articles found about the subject. Due to the fact that I am a student with only a small amount of experience with agile methodology acquired through a summer internship as a developer in the IT consultancy business. This experience is however not adequate to easily see the big picture that would have been desirable in order to fully understand the issues and challenges around coordination of teams in agile projects.

However, thanks to Torgeir Dingsøyr at SINTEF IKT, I have been granted access to extensive interviews (done by Torgeir himself) with Norwegian companies running large IT-projects. This is in the form of transcribed interviews with their statements on how they do their work in their projects.

It is hard to find the best way of implementing agile methods in projects, especially in projects that differ from the typical agile sweet spot. Furthermore, one method of work may suit one project, but it may be inappropriate for another project. Throughout this thesis, I hope to gather some tips and experiences from companies, articles and theory. It may be hard to test out the ideas in real life, but this thesis will hopefully provide at least some methods and ways to improve how people work in projects.

3 Theory

3.1 The introduction of agile methods

Agile methods in software development have been a hot topic over the last years. The term "Agile" was first introduced in 2001 with the "Agile Manifesto", so it could still be regarded as a relatively new way of managing projects. However, agile methods are used more and more, which is a clear sign of being useful. Studies show that an increasing number of organizations are interested in adopting the methodology (Dybå and Dingsøyr, 2009).

Agile methods were originally a reaction to the plan based ways of running a software development project. The plan-driven development methods like the waterfall model is a more traditional way of product development seen in other industries like mechanical engineering amongst other. Furthermore, software projects have also changed over the last decades. This is because the type of problems that software developers address has changed a lot the last few decades (Nerur et al., 2010). In the early days of computers, the target was to "increase efficiency by automating routine, structured tasks". The software was doing tasks like processing transactions and being Management Information Systems. According to Nerur et al. (2010), the number of stakeholders were quite few and had a "shared mindset". This made it unnecessary to have comprehensive involvement with stakeholders throughout the project. This means that plan-driven development easily could be suitable for these types of development projects.

However, as computers became cheaper and more common, the software became more complex. Many of the transaction based software mentioned developed multiple decades ago are still used today as core systems for banking, insurance and some government systems. In many cases, no one no longer knows how they work, while other newer systems runs on top, and depends on them. This is one of the reasons agile methods were introduced, because a lot of modern software projects depends on multiple subsystems, and they need to continually keep in touch with many stakeholders. An example of such a system is a tax-system. They need to communicate with banks, companies, employers, citizens and so on. This means there are a lot of stakeholders that the software development teams will have to stay in touch with during the project. If a tax system is to be developed it is more than likely that there will be changes throughout the project. The government may change

the way taxes are computed, sub-systems that the tax-system rely on may be changed and replaced, and demands and needs may change. When using the waterfall model in such cases, it will be cumbersome, if not impossible to complete the planning phase in the beginning of the project and then stick to this plan throughout the implementation phase. This means it will be necessary to change the plan continually during the project.

3.2 Predecessors of agile methods

Iterative and incremental methods were used in some form already in the 1950's according to an article by Larman and Basili (2003). Larman and Basili (2003) discusses some of the background history of why agile methods later appeared. They find many examples of situations where the traditional waterfall methods does not fit. They examine every decade and finds examples where people were describing methods that later would have been considered agile. Some years later Ernest Edmonds (1974) published a paper where he introduced an adaptive software development process. Additionally Tom Gilb started publishing a new concept he called evolutionary project management (EVO) in the 1970s (Larman and Basili, 2003) (Gilb, 2007). Gilb came up with a lot of methodology similar to agile, and amongst other, he is criticizing the heavyweight waterfall methods and claiming them to be obsolete.

In the 1990s, a serious wave of new lightweight methodologies was published. Amongst others: Unified Process in 1994/95 (Jacobson et al., 1999) (Larman and Basili, 2003), Dynamic System Development Method (DSDM) in 1994 (Stapleton, 1997), Scrum in 1995 (Sutherland and Schwaber, 1995), Crystal Clear (Cockburn, 2004) and Extreme Programming in 1996 (Beck, 1998), Adaptive Software Development (Highsmith, 1999) and Feature-Driven Development (Coad et al., 1999) (Palmer and Felsing, 2002) in 1997.

All of these frameworks were obviously made up before the agile manifesto was published, but all of them were a step in the same direction, and they all intended to bring more efficiency into software development processes.

3.3 The introduction of agile

The agile manifesto was published in February 2001 after a conference in Snowbird, Utah, with 17 software developers. This is when the term "agile" was first introduced. Many of the 17 developers had already contributed significantly on developing methods that from now on were considered to be "agile". The developers and authors that published scrum, crystal clear, extreme programming and adaptive software development, amongst other, were present at this conference. Indicating that all these methodologies and framework were introduced and implemented for the same reason, and with a common goal, namely "… uncovering better ways of developing software by doing it and helping others do it" (Beck et al., 2001).

The prioritization of values of agile manifesto are:

> *"**Individuals and interactions** over processes and tools*
>
> ***Working software** over comprehensive documentation*
>
> ***Customer collaboration** over contract negotiation*
>
> ***Responding to change** over following a plan"*
>
> (Beck et al., 2001)

Furthermore, Beck et al. (2001) explains how this prioritization values are to be interpreted:

> *"That is, while there is value in the items on the right, we value the items on the left more."*

After publishing the agile manifesto, the term "agile" was from now on considered as an umbrella term for the methodologies and frameworks that were developed and published a few years earlier (Dingsøyr et al., 2010a). The agile manifesto is therefore a common denominator for the concrete methodologies, and emphasizes the more abstract goals of agile methodology. The concrete methodologies are more specific in how to achieve these more abstract goals stated in the agile manifesto, which are more about values and beliefs.

In addition, the agile manifesto consist of 12 principles that further augment what the agile manifesto is prioritizing:

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

(Beck et al., 2001)

From the agile manifesto and these principles, it is not stated how to follow and archive these principles. For instance principle number 4: “Business people and developers must work together daily throughout the project” (Beck et al., 2001). The agile manifesto does not give any specific rules or guidelines on how they should meet, what they should discuss and so on. However, the manifesto is facilitating freedom for the stakeholders to choose whatever fits them best, as long as they work together on a daily basis.

3.4 Different agile software development methods

As mentioned earlier, a lot of new methods of software development was introduced in the 1990s, before the agile manifesto was published. These are more specific on how the dynamics in projects should be, and provides specific rules of communication and collaboration in order to become “agile”.

3.4.1 Scrum

One of the most popular methodologies within agile, is scrum. The abbreviated term “scrum” is originally from the sport of rugby, where “scrummage” refers to a formation of players where they are trying to take the ball (Oxford English Dictionary). In an agile product development context, the term was first used in an article called “The New New Product Development Game” by Takeuchi and Nonaka (1986). Takeuchi and Nonaka did a case study from companies in different industries like photocopy, printing and automotive business. They found that many companies had a very old fashioned product development process, that had the dynamic of a relay race (Takeuchi and Nonaka, 1986). The project was run sequentially from one phase to another, exactly like the waterfall method. The usual sequence they refer to was: “Concept development, feasibility testing, product design, development process, pilot production, and final production” (Takeuchi and Nonaka, 1986). This resulted in a handoff in the end of each phase where the project and the product was just handed over to a new group of employees. This might lead to problems later on, because the employees in the next phase needs to change aspects of the product in order to do their part of the work.

In the “rugby” approach Takeuchi and Nonaka suggests, a multidisciplinary team work together from start to finish. Instead of having predefined and structured stages, the project is carried out by overlapping phases where all the project member gets to use their knowledge from the beginning. The product design may easily be done in parallel with the feasibility testing, even though product design is performed after feasibility in the usual sequence mentioned above. When performing projects this way, the project members get more information for instance on why design choices are made the way they are, compared with a situation where a product prototype is just handed over with some documentation and the current team might overlook the previous teams intentions.

One of the important factors of the rugby approach is information sharing, because the project team members start with “zero information” (Takeuchi and Nonaka, 1986). According to Takeuchi and Nonaka, each member will share information continuously, and the result will be that the team is now working as a unit. Furthermore, Takeuchi and Nonaka claims that “at some point, the individual and the whole become inseparable” and that the team and the individual will get a overlapping rhythm, creating a pulse that “serves as the driving force and moves the team forward”.

Takeuchi and Nonaka (1986) is mostly referring to the “rugby approach”, and are only using the word “scrum” once. They do not specifically state what kinds of industries the rugby approach is suitable for, other than product development. So even though scrum is mainly something people associate with software, it certainly did not started out as a software specific approach, it could just as well be applied to other industries.

In 1995, Jeff Sutherland and Ken Schwaber presents a paper at the Business Object and Implementation Workshop as a part of OOPSLA conference in Austin, Texas. The paper they present describes the scrum methodology, and they are the first to refer to it as a single word, as opposed to Takeuchi and Nonaka, who described the rugby approach. They have both, independently of one another, been working with methods that would have been considered as scrum today. After the conference and the following years, they jointly worked together to merge and take the best practices from their experiences, the industry and their writings (Sutherland and Schwaber, 2010). Schwaber amongst others, founded Scrum Alliance, which offers courses and certifications.

According to The Scrum Guide (Sutherland and Schwaber, 2010), scrum is “Lightweight, simple to understand, difficult to master”. The scrum framework have defined different components it consist of. These components are teams, roles, events, artifacts and rules. The scrum team consist of team members with different roles:

- **The product owner** is the one person (cannot be a group) responsible for maximizing the value of the product being developed, and the work done by the development team. Additionally, the product owner is the one person responsible for the product backlog (which is an artifact, see below). The product owner can assign for instance

the developments team to help manage the product backlog, however, the product owner is the one responsible.

- **The development team** is the team of the professionals who work on delivering the increment of what is considered to be a finalized sub product at the end of a sprint. The team follows the principles of the agile manifesto (Beck et al., 2001), especially being self-organizing, and "… Give them the environment and support they need, and trust them to get the job done". The Scum Guide specifies, "no one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality". That is, the development team have flexibility to work the way they want in order to deliver increments.
- **The scrum master** is the person that have the responsibility to ensure that the team members understand scrum and agile methods, and that its principles are being followed. According to Sutherland and Schwaber (2010), the scrum master is a "servant-leader for the scrum team", meaning that the scrum master is helping communication between scrum team members and stakeholders. The scrum master is also responsible for keeping the events within the maximum time frame allocated.

Scrum have clearly defined events with fixed maximum duration in order to regulate meetings and cover the most common demands of communication. Scrum consist of these events:

- **The sprint** is the main event, and is the container of all the other events. It have a duration of maximum a month. In the end of the sprint, an finalized and releasable increment of the product is delivered.
- **Sprint planning** is done in the beginning of the sprint, and is performed by a joint effort by the entire scrum team, with a maximum duration of 8 hours. The team plan what items from the product backlog they are going to finish during the sprint, and how they will work perform the work.
- **Daily scrum** is performed in the beginning of each new day at work. The development team "synchronize" about what has happened since the previous daily scrum, and they create a plan for the next 24 hours. Daily scrum is often called "stand up meeting", because the team usually stands in a circle on the floor, and each member of the development team is answering the three questions:

  - What did I do yesterday (that helped meeting the sprint goal)
  - What will I do today (that will help meeting the sprint goal)
  - Is there any potential obstacles that may prevent me (from meeting the sprint goal)

  Having daily scrum enables the development team to quickly and easy get an overview of what the rest of the team are working on. If one of the team members are reporting any issues or potential obstacles, it may be beneficial that the rest of the team knowns about it, because there might be anyone else in the team who knows how to solve the problem. Those two team members might agree on meeting right after the standup meeting and solve the problem together, instead of discussing the problem while attending the standup meeting. This is important principle in order to keep the daily scrum within its maximum time frame of 15 minutes. The daily scrum is taking advantage of the principle from the agile manifesto stating that face-to-face is both the most effective and efficient way of communicating (Beck et al., 2001). As all projects starts with zero information, according to Takeuchi and Nonaka (1986), and the information is increasing along with the projects maturity, the daily scrum helps building up this knowledge (Sutherland and Schwaber, 2010).

- **Sprint review** is held at the end of sprint, and the team inspect the increment. The product owner have to decide which items from the product backlog that can be marked as “done”, and which items that are not approved as “done”. The development team demonstrates the work that has been marked as “done”. The group also have to discuss if demands, requirements and expectations have changed during the sprint, and based what they come up with here, they might change the product backlog by adding, removing or changing items. Additionally the prioritization of its items may change.
- **Sprint retrospective** is an event held after the sprint review. As sprint review focus on the product increment delivered, the retrospective, or just “retro”, is focusing on the team itself, and how they worked together as a team. They look into factors as people, relationships, processes and tools. The team may divide the feedback into three categories: What worked well, what did not work well, and how can we improve what did not work well. The goal of the retrospective is to be more efficient when working on future sprints. This event is directly addressing the twelfth principle

in the agile manifesto, about reflecting over how to increase the effectiveness of the team.

Scrum have artifacts that represents value or work. These are according to The Scrum Guide (Sutherland and Schwaber, 2010) made to “provide transparency and opportunities for inspection and adaption”. This means everyone in the scrum team can see them in order to bring all information in the project organized and available for everyone. This is an important principle of scrum, that information is available and accessible for everyone at all times. The artifacts are:

- **Product backlog** is a list of everything that are needed in the product, and is sort of a requirement specification for the product being developed. A notable difference in agile projects compared to project with the waterfall method, is that the product backlog is changing constantly, while the waterfall method may require a change order before if the requirement specification is changed. The product backlog is never complete. This is due to the second of the twelve principles in the agile manifesto, saying that changes are always welcome, even late in the development project. This might sound cumbersome, because changes late in the project is likely to decrease efficiency, however, the agile manifesto states that if a change is what it takes to make the customer more satisfied and have a larger competitive advantage, it is worth doing the change.
  Furthermore, the product backlog describes features, requirements, enhancements and fixes to the product under development. All the items in the product backlog have a description, an order, an estimate over required work, and a value.
- **Sprint backlog** contains a subset of the product backlog, with the product backlog items selected for the current sprint. During a sprint, all the items in the sprint backlog are scheduled to be marked as “done” when the sprint is over. In addition to just linking to the product backlog items selected, the sprint backlog also consist of a details on how changes in progress can be understood in the daily scrum. As with the product backlog, the sprint backlog changes continuously throughout the sprint. However, only the members of the development team are allowed to do changes to it. Changes that typically occur is when they realize one item in the sprint backlog depends on something from the product backlog that is not marked as “done” yet.

The item from the product backlog may then get a higher priority and will then be added to the sprint backlog. If the product owner wants to change something, it has to be made through the product backlog first, then the development team will have to do change the sprint backlog accordingly to adapt to the changes.

3.4.2 Scrum of scrums

In large projects, there is often more than one development team. This introduces a new set of challenges, and standard use of scrum might not be adequate, due to the fact that there is no rules of how team communicates with each other. It would have been cumbersome to gather all the developers from all the teams in one large standup meeting, because it would be too much information for everyone to handle. In addition it will be very time consuming, and it will be impossible to keep it within the 15 minute time frame. Thus, it will be impossible to define a large scale project as “agile” that way.

One way of dealing with large projects with multiple teams, is something called “scrum of scrums” (Sutherland, 2001). This is done by dividing the group of developers into sub-teams with around 10 developers. Each of these sub-teams will then assign one of their team members to be an ambassador for that sub-team. The ambassador may be considered the sub-teams scrum master, and this person could contribute technically, depending on the project. Each of the sub-teams are supposed to have their own separate daily scum, and the ambassador will afterwards attend the scrum of scrums, reporting on behalf of their sub-team. In addition to the three typical questions answered in the standup meeting, the sun-teams ambassadors might as well reflect on whether they are going to do something that may interfere and put something in another sub-teams way. This way they ensure that sub-teams are working consistently when they are developing components that are tightly related to the work in progress done by another sub-team. When the scrum of scrum meeting reveals that two or more sub-team are working on something related, the ambassadors and their respective developer teams should preferably arrange a new short meeting afterwards clearing things up, and divide the work in proper way. This is important in order to reduce inconsistency when the individual increments are to be used together and integrated later in the project. This is according to the agile manifesto done best by face-to-face meetings (Beck et al., 2001).

The ambassadors will typically report only the highlights of the standup meeting held with the sub-team. The scrum of scrum meeting might provide information that the ambassador should report back to the sub-team. An example can be if two ambassadors, during the scrum of scrum, is reporting the same types of problems from two different sub-teams, they might arrange a new meeting with the people experiencing the problem. They could then solve the problem together across the different sub-teams.

3.4.3 Crystal methods

The crystal methods was developed by Alistair Cockburn in the 1990s. They methods are also known under the name crystal family, because it consist of different methods for different purposes. The family members are divided into different colors depending on size of the project. It is also divided into one of four categories depending on what is the critical success factor like life, essential money, discretionary money or comfort. Those two dimensions, size and criticality, form a matrix where the different methods are arranged. When starting up a project, the project managers are supposed to choose an appropriate method, and apply it. As larger projects are likely to require more coordination as well as more formal methods (Abrahamsson et al., 2010). In addition, crystal methods are flexible when it comes to facilitating for integration of other practices like extreme programming and scrum, in order to combine the best of all the methods.

3.5 Traditional coordination of teams

Many projects have a large number of teams whether they are using agile methods or not. As the number of teams increase, the need for coordination between them is also increasing, in order to be both efficient and effective. This section will review various methods of coordinate multiple teams in projects.

3.5.1 Communities of practice

Community of practice is a term introduced by anthropologists Lave and Wenger (1991), in their book "Situated Learning: Legitimate Peripheral Participation". A community of practice is according to Wenger et al. (2002): "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 an ongoing basis". A variety of industries and organizations have already applied communities of practice in order to coordinate and improve effectiveness and efficiency (Wenger et al., 2002). Additionally they claim that "every organization and industry has its own history of practice-based communities, whether formally recognized or not", meaning that communities of practice are forming almost automatically. They explain this by referring to the fact that all automakers in the USA is based in Detroit, and that the worlds technology industry is centered in Silicon Valley, just outside of San Francisco.

There are, according to Paasivaara and Lassenius (2014), three important characteristics that makes them different from other types of communities. These characteristics or aspects are:

- **Domain** – specifies the area of interest the members are working with and sharing, collaborating and creating information about.
- **Community** – means that the members are a part of, and contributes to joint activities, build relationships, and share information about the domain with each other.
- **Practice** – over time, people working within the same community are likely to acquire a shared set of tools and resources for addressing and solving problems in the domain they are working on.

Communities of practice may have different types of relationships to the organization, according to Wenger et al. (2002) and Paasivaara and Lassenius (2014). The different types

are unrecognized, bootlegged, legitimized, supported and institutionalized. As mentioned earlier, communities of practice will usually build up automatically, but a community may become one of the different types depending on how the organization support, recognizes, and takes advantage of them.

Communities are also subject to an evolution, they grow and evolve and may cease to exist eventually. They might as well be transformed, renamed and/or merged with other communities of practice as the demand of a particular community change, or if a community is evolving in the wrong direction. There are five stages of development a community of practice may go through according to literature on (Paasivaara and Lassenius, 2014):

- **Potential** – Denoting when there is not yet a community, but a group of people is sharing a common interest for a topic. From this point it is critical to find a sufficient ground for the group, and realize that there is value in a potential community of practice. That is, they will have to acknowledge the potential value of having meetings, sharing information and knowledge among coworkers and solving problems together. It will be beneficial to assign a person to become a community coordinator, in order to keep the community alive.
- **Coalescing** – At this stage, the community is aware of what the organization may offer regarding its domain. Events are being held, but it is important to deliver value immediately to avoid the community to lose interest.
- **Maturing** – The stage when the community have reached stability, and proved its demand and necessity. The community of practice should now focus on clarifying their focus, roles and boundaries.
- **Stewardship** – At this stage it’s all about maintaining its position, and continue at serving its purpose.
- **Transformation** – This stage is denoting the situation where the community becomes discontinued, or it is transformed in to a new and different structure. The community can also become institutionalized in the organization, for instance as a separate department.

The use of communities of practice have been proposed as a possible solution of how to effectively and efficiently share knowledge between organizational units, especially in organizations implementing large scale agile software development (Paasivaara and Lassenius, 2014). There are not a lot of empirical studies on how to use communities of practice within agile software development. However, according to Paasivaara and Lassenius (2014), Nokia have used a kind of communities of practice to solve issues between teams in multi-team projects. Nokia is also stating that communities of practice might be a suitable tool for organizations that are implementing large-scale agile development.

The way Paasivaara and Lassenius (2014) suggests using communities of practice is to have different types communities depending on its goal. For instance, an organization may have “feature communities of practice” with the purpose of coordinating between multiple teams working on the same feature. Another example is “coaching community of practice” that are a community where the members can discuss implementation issues regarding the transformation into agile development. This community of practice is in many ways similar to the sprint retrospective known from scrum.

3.6 Coordination of teams in agile projects

Based on the statement mentioned earlier, that "every organization and industry has its own history of practice-based communities…" (Wenger et al., 2002), it becomes clear that communities of practice is not something that relates to any specific way of working for instance like agile methods. It is a quite general term and a way of working that could be applied to any project, whether agile or not. There are a lot of research around communities of practice (Wenger, 1999), but the combination of software development (and especially agile methods) and communities of practice have not received much attention literature (Paasivaara and Lassenius, 2014).

Furthermore, as more and more organizations are taking the step and starting to implement agile methods (Dybå and Dingsøyr, 2009), Paasivaara and Lassenius (2014) are emphasizing that many organization does not know how to handle the transformation over to large scale agile methods. Especially large organizations do have a lot of "institutionalized processes and organizational structures that provide a poor fit with agile development (Paasivaara and Lassenius, 2014).

However, there are some methods and expansions to existing agile methods that seek to improve how teams are coordinated in large projects. Some are made specifically to certain existing methods, like scrum, while other may be suitable for any agile methodology.

3.6.1 Software tool support for agile organizations

Chau and Maurer (2004) suggest using communities of practice in combination with their experience factory concept, and the use of wiki based knowledge management tools. By using their approach, they aim to share knowledge across teams without moving people between the different teams, or reshuffling the teams.

An experience factory is "an organizational unit that gathers experiences, makes them accessible to others (e.g. in the form of 'best practices') and provides line units with support in adopting these best practices." (Chau and Maurer, 2004). The main difference between communities of practice and experience factories is that experience factories are a more standalone organizational unit, where its members may not be part of any of the teams that are working on the project. This might be problematic, because it may be a gap between the

members of the experience factory and the members of the actual development team. Thus, the members of the experience factory might be too theoretical, and even be referred to as living in an ivory tower. Chau and Maurer (2004) also states that experience factories often tends to prefer a controlled learning process in addition to focusing on explicit knowledge over tacit knowledge, and by that contradicting the agile principle of autonomous and self-organizing teams. It is also contradicts the first statement in the agile manifesto, “Individuals and interactions over processes and tools” (Beck et al., 2001).

Nevertheless, Chau and Maurer (2004) suggests using a combination of both experience factory and communities of practice. They examine two software tools, called EB and MASE, which are integrated with each other as well. MASE is a tool for registering tasks during a project, and each task is, amongst other registered with a description, the name of the responsible developer, and a link to an “experience base”, that may be clicked and then opened in EB. From here, the developer have access to a knowledge base that will support the developer on achieving the task assigned to him. EB is more specifically a tool that “allows users to construct models of common tasks” (Chau and Maurer, 2004). The content of EB might be possible for everyone in the organization to edit like a wiki. This way the database will stay up to date, as people get new experiences while running projects. EB might store tips and tricks, as well as warning regarding pitfall. A typical scenario is when a developer is picking an item from the sprint backlog, he may check the link into EB, and get a quick overview regarding what he should pay extra attention to when working on the item. This may help people working in the organization to be more efficient and effective. Especially when multiple teams are working on peripheral components that depends on each other, this might avoid mismatches and inconsistencies, and will help getting it right already the first time.

3.6.2 Large-Scale scrum

In an article by Larman and Vodde (2013), they present two frameworks for large-scale scrum. The first framework, “Framework-1”, is intended for projects consisting of up to 10 teams with about seven persons in each of them. “Framework-2” is for projects with any larger size, and up to some thousands of people. The transition between framework-1 and framework-2 is not very sharp, and the real tipping point depends on other factors as well, in addition to just the number of teams. One of these other factors are when the project owner no longer are able to interact with the teams in an effective manner. There might be many reasons to why the product owner are not able to interact with all the teams. The product backlog may be too big, and as previously mentioned the product owner is the single person responsible for the product backlog (Sutherland and Schwaber, 2010). Another reason may be that the product owner cannot balance the focus on both external and internal interactions. This again might be caused by too much complexity in the scope, or just the number of teams. Thus, it is important to evaluate what kind of project it is before choosing one of the frameworks.

A group of people might assist the product owner, even though the product owner himself is responsible for the product backlog. The framework also suggest that the product owner might delegate work back to the development teams, and if appropriate, the developer team members may get in touch with the stakeholders themselves, instead of having the product owner to do it. This ties back to the principle from the agile manifesto, that “the best architectures, requirements, and designs emerge from **self-organizing teams**” (Beck et al., 2001). Furthermore, the framework emphasizes that the product owner “does not need to be involved in low-level details – they should be able to focus on true product management” (Larman and Vodde, 2013).

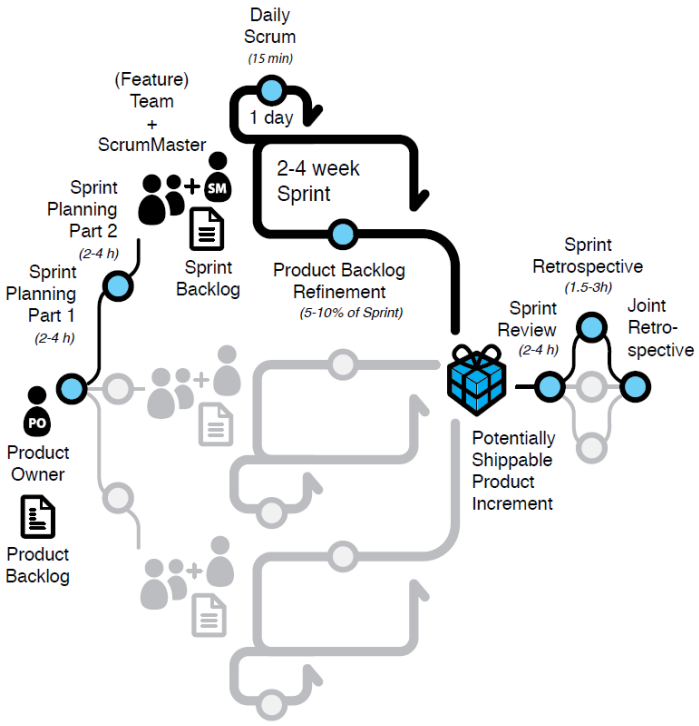

Figure 4 – Illustration of Framework-1 (Larman and Vodde, 2013)

There are many similarities between normal scrum and the frameworks, but there are obviously some differences. First of all, each development team have their own sprint backlog and sprint goal. When it comes to sprint planning, the framework arranges two rather than just one as in normal scrum (see Figure 4). Sprint planning part 1 are held with the product owner as usual, but with only two members from each of the development teams. They pick product backlog items for each teams individual sprint backlog. In addition, they should identify if there are any dependencies between the sprint backlog items across teams. If so, they will have to discuss how they are going to coordinate the work around this. Next step is the sprint planning part 2, where each individual team is having their own sprint planning meeting. In some cases, for instance where two (or more) teams are working on backlog items that depends on each other across teams, the other team (or teams) may be invited as observers. The observers may as well come up with suggestions on how to solve coordination issues.

When it comes to the daily scrum, these are performed independently in each team, however, a team may invite members of other teams as observers if they are going to discuss something that might be relevant for people outside the team. The framework also

suggests having scrum of scrum several times a week, by sending an ambassador from each development team as described in section 3.4.2 and by Sutherland (2001).

Halfway into the duration of the sprint, it is also suggested to do product backlog refinement, where all team members are meeting up, and preparing for future sprints. This way they all get an overview over the backlog items that are coming up in future sprints. In addition, they get the opportunity to suggest changes.

During the sprint, the framework suggest an optional additional event called “in-sprint backlog item inspection”, where the product owner and/or other stakeholders are invited for a demonstration of finished sprint backlog items, in order to reduce the amount of inspection needed in the sprint review. This is an opportunity to get feedback early on, and reducing the risk of making mistakes that the product owner otherwise would not have found until the sprint review in the end of the sprint. Additionally, this might allow the product owner to get a continuing impression on how the project evolve, and ensuring that the project is on the right track.

Furthermore, the framework suggest having a sprint review with only two persons from each team, together with the product owner and other stakeholders. As the sprint review is meant to be a demonstration of the work that is done, combined with the fact that there are many teams, it is suggested that the representatives brings their own computer and line up in a large room. This way the product owner and the stakeholders can quickly and easily walk from one team to another while inspecting and hopefully approving the work done in the sprint. The sprint retrospective are held individually in each team, but additionally they may arrange an extra joint retrospective, where the scrum master and one representative from each development team discusses their conclusions.

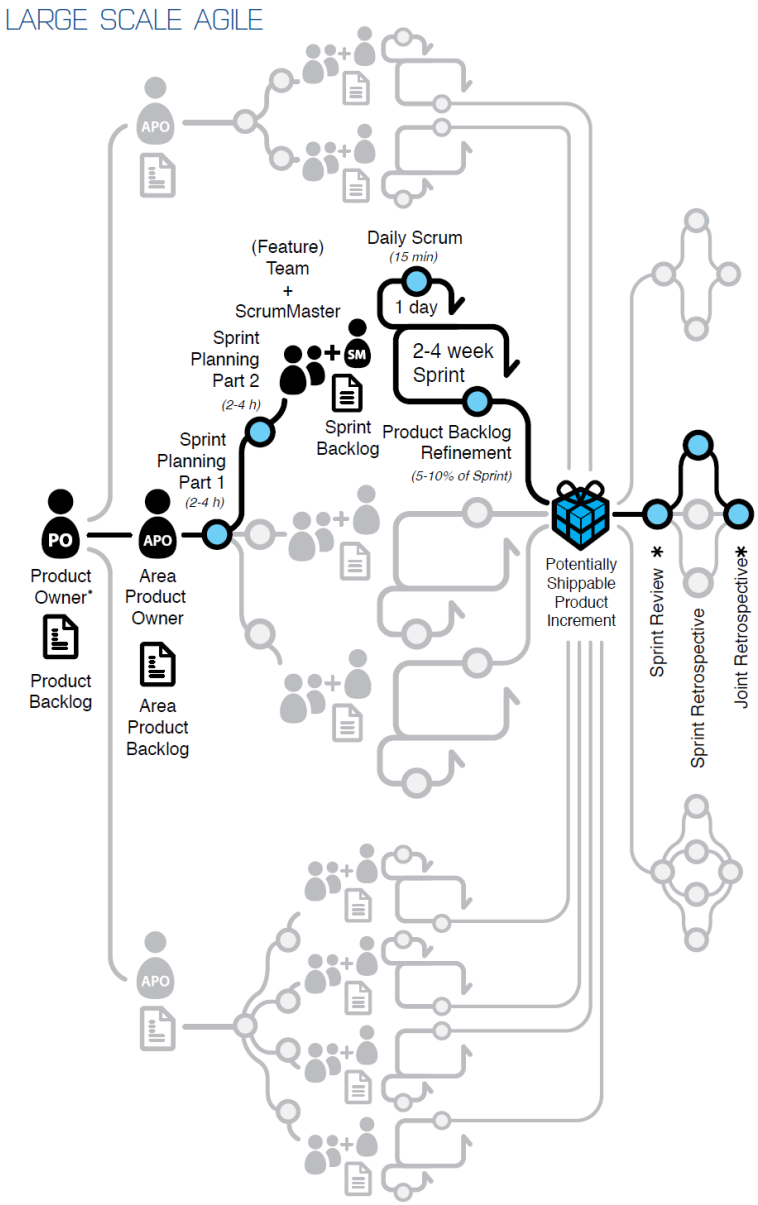

Figure 5 – Illustration of Framework-2 (Larman and Vodde, 2013)

When it comes to the difference between the two frameworks, framework-1 and framework-2, other than the number of teams, the most significant is the introduction of area product owners in framework-2 (see Figure 5). When there are a vast number of teams, the teams are divided into areas, where each area has their own area product owner. The areas are divided based on the customer requirements area of the final product. Examples of a requirement areas are network protocols area, or performance area (Larman and Vodde, 2013). Framework-2 also introduces pre-sprint meetings with all the area product owners, in order for them to coordinate before the upcoming sprint planning. The area product owners

are in framework-2 considered to be a “product owner team”, but there is still one person who is the “product owner” who have the responsibility over the area product owners.

3.6.3 Kanban and Scrum-ban

When using agile methods it is common to use some sort of tools to store and organize the product backlog, sprint backlog, and whom the different items of those backlogs are assigned to. One way of doing this is using Kanban, or Scrum-ban that is Kanban for projects using scrum.

Kanban is a Japanese word that means “signal card” in English (Anderson, 2010). The term originates from “Toyata Production System”, well known for being a highly efficient production system. When utilizing Kanban in a project, there are one or more whiteboards containing small cards representing work packages or product backlog items. The boards have multiple rows and columns, where the cards are systemized. The usual columns are “to do”, “doing” and “done”. The development team is then picking cards from the “to do”-column, and subsequently moving them over to “doing” and finally to “done”. They might as well write their name on the card when they pick it from the “to do”-column, in order for the rest of the project team to know who is doing what. Furthermore, they avoid two people from doing the same. Additionally it is easy to track the progress of the project. The main difference of Kanban and scrum-ban is that scrum-ban is only showing the current sprint, while Kanban is usually showing the whole project. However, Kanban is a not tied specifically to software projects, and when using it in agile software projects the difference between Kanban and scrum-ban is not very big, due to the fact that agile methods are based on iterations (like sprints). The terms Kanban and scrum-ban are to some extent used interchangeably in the software development context.

In addition to the columns mentioned, it is also common to have board with information everyone can benefit from. For instance, it may be convenient to have a board for issues they are experiencing. If a developer is struggling to fulfill a task, they can move it to a “issue” board or a “software bug” board. This way the rest of the team can have a look occasionally. Hopefully, someone else in the project team know something that the other developer did not know, and they may help each other out.

3.6.4 Software systems for coordinating

A common addition or alternative to Kanban and scrum-ban, is the use of issue tracking systems. These are software systems representing the cards and boards of Kanban and scrum-ban. These systems are often highly flexible and can be customized for a lot of use-cases. If the project is using scrum and consist of multiple teams, the system lets project members assign backlog items to teams or specific persons, and project owner, stakeholders and other project team members can instantly watch the progress of the project. These software might be more suitable in very large projects, because the number of backlog items is very high, and using card will be to complex. The software allows to prioritize and link backlog items that depends on each other. Additionally, it is easier to change and edit backlog items when using software, compared to editing paper cards.

3.6.5 Software architecture

So far, the coordination methods have been concerned around communication. However, another more abstract and maybe indirect way of coordination is the use of software architecture. When building software products, it is essential to organize how the systems are built up, and by which components it consist of and how these components communicate with each other. In agile projects, the teams are usually supposed to be cross-functional, and therefore there are not common to have a dedicated architect in the team. However, holding all development team members equally responsible for the architecture might increase the risk of having the project drift out of control, and end up having incompatible components (Eckstein, 2014). According to Nord et al. (2014), “The architecture of a software system is a metaphor, analogous to the architecture of a building” (Nord et al., 2014). Furthermore, architecture in the software context defines the high-level structure, and is therefore a good way of organizing and modelling the work, in order to visualize and fully understand how it is linked together.

Eckstein (2014) suggests different models as solutions for how to focus more on software architecture, without compromising the agile principles. The appropriate model for handling the architecture should be chosen depending on the amount of changes, and the uncertainty

of the project. In a complex system, Eckstein (2014) suggest that the project may have a so called “Technical service team” that is a separate team that have the responsibility of the architecture. This team might as well be the only team in the beginning of the project, and then they will define the skeleton of the system. When the project is escalating by increasing the number of project team members, the technical service team might split up and divide themselves evenly into the cross-functional development teams.

In later phases of the project, or in projects with less complexity in terms of changes and uncertainty, another solution may be a “technical consultant team”. This team is a flexible team with relatively few members, and the members may join the developer teams for one sprint or iteration at the time, depending on the demand. The team might also just function as a small group of consultant that may be contacted by development team members when they need advice on how to keep their work consistent with the rest of the project.

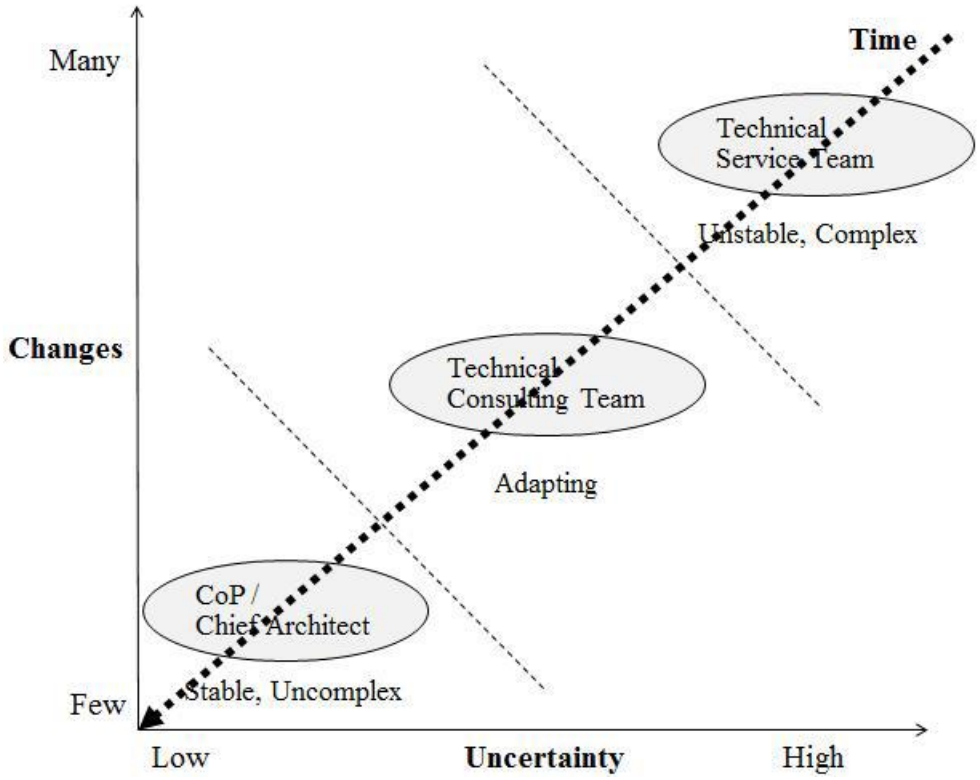

Figure 6 – Illustration of the different cases where the various team structures suit best (Eckstein, 2014)

When the project is maturing, and if the projects architecture is not that complex any more, the project may only keep an “architecture community of practice” to maintain and regulate the architecture. Most of the members of either the technical service team or the technical consultant team may work as regular development team members from now on, but some might still participate in the architectural community of practice beside. A chief architect or an “architectural owner” may manage this community, and this person will then be the only person solely working with architecture. In Figure 6, the black arrow pointing to the axis origin is showing a typical evolution of a projects declining demand of architecture in the end of the project.

4 Results

4.1 Case study

Chief Scientist Torgeir Dingsøyr at SINTEF ICT, have carried out extensive interviews with multiple Norwegian companies in the IT consultant business that are running large scale software projects using agile methodology. They are provided to me as transcribed interviews, done by Dingsøyr meeting face-to-face with the companies.

The transcribed interviews are anonymized, yet confidential and access to them are therefore subject to a confidentiality contract. All of the interviews are done in Norwegian, and any quotations are translated to English by the author of this thesis.

The three interviews were done with three different organizations, where project managers from each organization are asked a series of questions, in what seems to be a semi-structured way, as new questions are brought up along the way, in addition to the questions prepared in advance in the interview guide, also done by Torgeir Dingsøyr (see section 8.1 Appendix). The questions are about different aspects of project management, like success factors, estimation, and working methods. Not all of those topics are relevant in team coordination, and are therefore not presented in this thesis. The relevant conversation topics used for this thesis is what they tell about agile methods and coordination of teams. Therefore, one of the interviews were omitted, due to lower relevancy.

The three organizations are anonymized and given generic names like “Alpha”, “Beta” and “Gamma”, and will be presented separately.

4.1.1 Organization “Alpha”

Two project managers are interviewed from “Alpha”, and are denoted AP1 and AP2. They were interviewed about a project that was ran by Alpha jointly with some other organizations. Alpha and the other organizations were a group of software vendors, supplying an external customer. The two project managers were a team of multiple project managers in this particular project, that were using scrum.

AP2 have a technical background and started as a developer, and have then been promoted to team manager, project manager and program manager. He tells that he did not have any experience with agile methods at all, until recently, and was very sceptic to it in the beginning. He says the skepticism was related to the flexibility, more specific a fear of too much flexibility for complex projects. However, he turned out to be quite convinced after a while, and stated that he could imagine using something other than agile methods in the future. On the other hand, he says that even though agile methods have provide good results, it is still absolutely possible to run an agile project into complete failure.

AP1 have some of the same background as AP2. He went from being a developer to a technical manager, technical architect and finally project manager for multiple teams. He did not either have experience with agile recently.

Even though both project manager only have a short experience with agile methods, they both see the value of it. It is nevertheless, a bit surprising they did not seem to have any formal "education" on agile, or any certifications. They both seem to have learned agile methods by experience from doing it in projects. Based their lack of formal education, and the statement that it is fully possible to run an agile project into complete failure, they indirectly verifies the statement from the definition of scrum. Namely that "scrum is lightweight, **simple to understand, difficult to master**" (Sutherland and Schwaber, 2010).

Furthermore, AP2 emphasizes the importance of anchoring agile methods from the top management. In addition, he states: "Agile does not have that much to do with system development, […] it is more concerned as an approach to run projects, and I think the most important part is to put the business oriented stuff in the driver's seat in order to decide and anticipate what is coming out in the end of the project". This support the principle from the agile manifesto, stating that "Business people and developers must work together daily throughout the project" (Beck et al., 2001). In addition, AP2 believes that a traditional waterfall project usually does not give the same access to the business aspects of a project. By using agile methods, he believes the project team does not require that much of understanding of the domain. Agile methods may give each developer a possibility to talk to the project owner (or any of his representatives), and that way they are not just "left alone"

with a requirements specification, and the developers have a limited possibility to speak with the customer and project owner.

The project AP1 and AP2 was interviewed about was at some point consisting of 13 teams. Prior to each delivery, the teams were gathered in a meeting where a drawing of the enterprise architecture were shown. They zoomed in on different parts, showing and highlighting which parts each team are going to work on for the next period of time. This way, they got an introduction to how the system was interrelated and how the different parts was supposed to fit together.

AP1 emphasizes the importance of keeping the team members satisfied, and listen to those who wants to change what they work with, for instance they might rotate and switch between if desired. In a large project, there are always new tasks emerging, and those who wants some variation should get that possibility. Additionally, if a person have been in a project for a while, they should be given the opportunity to become the scrum master in a team for a sprint or two. Furthermore, if the interaction between the teams are well functioning, they should be left alone, being self-organizing, otherwise the managers should do some rotations and switch some roles in the teams.

The project also had a functional architect, who worked with the persons with the functional responsibility and technical architects from the development teams. Although it is not described very detailed in the transcribed interviews, it seems like they were using a kind of a technical consultant team, described earlier in section 3.6.5 about architecture. This group worked closely with the customer of the product. They had a standup meetings, and meetings where they discussed progress and issues that emerged.

AP1 further explains that they were using scrum of scrums, but also something he called scrum-of-scrum-of-scrum, where development leader, project managers for the other software vendors, the customer’s test manager, the customer’s project managers, and the business architect were attending twice a week. They were having a general status report, and discussing obstacles, risks, if they were on schedule and so on.

In conclusion, AP2 emphasizes the role of the architecture as a driver for the solution description. Additionally, the close relationship with the customer, through the group of

functional architect, functional responsible and technical architects, were one of the drivers for success in the project. Furthermore, AP2 tells that if he were to manage a waterfall project in the future, he would still use elements from scrum, like the daily scrum event, and keep the same amount of cooperation with the customer.

4.1.2 Organization "Beta"

This interview is with project manager BP1 and BP2, and functional architect BF1, about a large public IT-project, involving multiple suppliers like in the previous interview.

BP2 is discussing the use of scrum in large projects. He states that the default scrum methodology is not entirely suitable "as is" in large projects. This is because default scrum is tailored for the agile "sweet spot". However, in large projects, the project owner is not an omniscient person capable of answering everything.

When it comes to team coordination the project managers gave the teams "freedom under responsibility". The project managers wanted autonomous teams that were able to figure out things on their own. This way, "they get a stronger ownership to the work they do", BP1 states. "It is more fun working in a team where you feel like it is the team itself who find out how we are going to solve this case, instead of letting anyone who just happen to be the boss or something tells us how to do it" BP1 continues. However, in cases where they did not feel the teams were working optimally, the issue on how they should improve it was raised. If they just went in and gave commands, the team would no longer have been considered autonomous.

When it comes to coming up with solutions on how to do things, BP1 emphasizing the importance of letting the team discuss how to do it. Not only to come up with the best possible solution, but also ensuring everyone in the team is getting a common understanding of the problem itself, and how they decide to solve it.

BF1 (functional architect) tells that he, in cooperation with technical architect and product owner, usually every day, used to "play" with the product backlog and look at dependencies and try to find out which items suited which teams.

BP1 uses the metaphor a “buffet lunch” about the product backlog, where the teams had the ability to pick thing to “eat” from left of the table first. However, even though teams are supposed to be self-organizing and autonomous, they cannot be allowed to pick the wrong items.

Furthermore, BP1 and BP2 jointly tells about the usage of mini-demonstrations of preliminary versions, similarly to the in-sprint backlog inspection from the large scale scrum framework by Larman and Vodde (2013). BP1 mention that for instance, they might sort out issues like “we want a drop-down menu rather than a free text search box”.

5 Discussion

Throughout the thesis, different ways of coordinating projects have been presented, in addition to a review of transcribed interviews conducted by Torgeir Dingsøyr. Especially in the interviews, the methods used for managing projects and coordinating teams were not following any specific frameworks, other than basic version of scrum. At least it is not mentioned that the methods used were something other than an "acquired over time" way of working. That is, none of the project managers mentions any literature methodologies backing up their way of managing the projects. However, some of these "methods", or at least way of working, seems to fit some of the suggested methodologies for large-scale agile mentioned in literature. This is could be both positive and negative. Positive because it confirms the methods found in literature as legitimate and well-functioning way of working, as the project managers interviewed felt satisfied about how they worked. However, on the negative side, they did seem to base all their choices on personal opinions and experiences. They did not necessarily have much formal knowledge on scaled agile methods, which could be a problem if they do not know, for what situations the methods suits. Furthermore, they might not know the limitations of the methods they are using. For instance, the different methods defined by Eckstein (2014), that depends on the uncertainty and amount of changes in the project.

The coordination methods presented earlier are written by authors who claim to have used their approaches in the real world. Most of the articles describes how they ended up doing it. For instance, the scaling agile framework by Larman and Vodde (2013), are written based on how they (Larman and Vodde), have worked as consultants with clients, and helped the clients implementing scaled agile environments, with up to thousands of people. Amongst the clients are the telecom company Ericsson and Bank of America Merrill Lynch. The author of the article "Architecture in Large Scale Agile Development", Jutta Eckstein (2014), have also been used all the approaches mentioned, like technical consultant team, technical service teams, and communities of practice. She therefore claims the approaches to be well proven in praxis. This is also to some extent proven by the interviews done by Torgeir Dingsøyr, using a group of architects working in the different development teams.

Additionally, the article by Paasivaara and Lassenius (2014), describes the use of communities of practice in a research and development organization within the telecom

company Ericsson. In this case, the organization was transforming from the traditional waterfall approach, into a new approach, utilizing agile and lean principles. Ericsson was performing excellent in running waterfall projects (Paasivaara and Lassenius, 2014). However, the background for the transformation was to decrease lead-time, because of the fast moving industry with tough competition. This was therefore a step made in order to regain competitive advantage. They used scrum, but in addition, they used communities of practice to enable cross team coordination, learning and information exchange. Not surprisingly, the transformation needed some time to establish a smooth pattern. One of the problems they experienced was that the motivation and the desired outcome of the communities of practice was not always the same amongst its members. Therefore, some communities of practice were discontinued and just dissolved. However, in the end, they managed to evolve the way of using the communities of practice, and the communities became a key to the successful transformation into agile. Bases on the experience from interviews, Paasivaara and Lassenius (2014) identified eight characteristics of how to successfully take advantage of communities of practice. These can be summarized quickly like this:

1. Having an interesting topic, every single meeting should give benefits for each of the participants' daily work.
2. It should have a leader to manage the meetings, and come up with an agenda.
3. The agenda should be distributed before every meeting.
4. The community of practice should have a decision making authority on the subject discussed.
5. The community of practice "needs a community around it, which is open and transparent".
6. The community of practice needs tools that can store and make the information and decisions available, for instance a wiki page.
7. It should have a "suitable meeting rhythm".
8. A community of practice should support and enable "cross-site participation", in order for everyone in the organization to benefit from it.

One of the slightly surprising finds throughout the thesis was the apparent compatibility between the different ways of coordinating projects. By applying any of them, the others are not foreclosed, they can be applied as well in combination with the rest of them. Obviously, the methods within a framework, for instance the scaled scrum framework by Larman and Vodde (2013), the project managers have to choose between framework-1 or framework-2, but either of these can be combined with communities of practice, or with one of the architecture solutions suggested by Eckstein (2014).

Furthermore, the introduction of agile have brought in new ways of working, and an apparent issue when it comes to scaling up agile methods is the fact that the teams are supposed to be cross-functional and flexible when it comes to performing tasks. In addition, the development teams are supposed to be working in a functional, and feature oriented manner. Meaning that an item from the backlog often involves implementing a complete, independent feature. This is for instance backed up by one of the principles from the agile manifesto, stating that "working software is the primary measure of progress" (Beck et al., 2001). This might in general look like a solely good idea, however it require the software team to have a more deeper knowledge to the entire stack of technology in use (Bick et al., 2014).

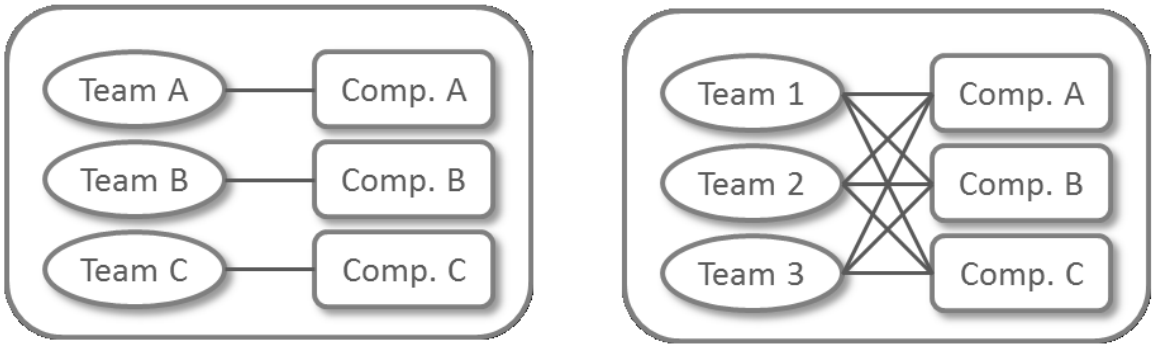

Figure 7 – Illustration of different ways of assigning tasks (Bick et al., 2014)

In traditional waterfall projects, it was more common to let teams work with one component each as in the left box in **Feil! Fant ikke referansekilden.**. When using agile methods and

being feature oriented, they lose some of the ownership to their components. This may cause the teams to risk conflicts and interference between the work of the different teams, along with misalignments of the work being done. According to Bick et al. (2014), “the resulting effects on the organizational and product structure, of this are still widely unknown”. However, as earlier mentioned, to deal with these issues, using communities of practice across the teams, in addition or alternatively the usage of the architectural approaches by Eckstein (2014) may help overcoming these challenges.

6 Conclusion

Agile methods have grown from being a way of managing small projects, to becoming adopted and scaled up to every level of large organizations. Even though agile methods, and especially large scale agile, still are considered relatively new methods, it is obvious that they are here to stay, the interest for them are still growing further.

6.1 RQ1: How is coordination between agile teams performed?

Throughout this thesis, various ways of coordination have been identified through literature. The agile manifesto clearly states that development teams are supposed to be self-organized and autonomous within themselves (Beck et al., 2001), however, when it comes to how the communication and coordination are being done *between* teams, there are not any clear rules defined from the agile manifesto. Nevertheless, through literature and the interviews from Torgeir Dingsøyr, it is clear that main ways of coordinating is by using communities of practice, or using some group of people dedicated to architecture. In addition, it is usual to use scrum of scrums, approximately every other day, in order to handle issues that emerge.

There are clearly not one universal way of coordinating teams, as projects are unique both by nature and by definition. This calls for methods that are adaptable and flexible, but it is also crucial to evaluate what kind of project one are dealing with in order to choose the right method.

6.2 RQ2: What are the alternatives for coordination between agile teams?

The main alternatives identified for coordination of agile teams are:

- Communities of practice
- Using architectural teams
- Scrum of scrum
- Scaled scrum framework (Larman and Vodde, 2013)

However, even though denoted here as “alternatives”, it does not mean each of them are to be chosen exclusively. That is, you do not have to choose only one of them. They may easily be combined with each other, and they have different areas they might be appropriate for.

Communities of practice are suitable for projects where there is desirable to let regular development team members meet across teams to discuss issues that concerns all of the members in the community of practice.

Using architectural teams are optimal when the coordination issues are mostly concerned around the structural aspects of the product being developed. The team of architects might be spread out in each of the teams assisting the developers on how they should align their solutions to the work done by the rest of the development teams.

The scaled scrum framework and scrum of scrums are mostly concerned about managing and creating an organizational structure for the entire project consisting of a large amount of project members. The framework is mainly focusing on how to be able to use the events, artifacts and roles known from scrum in a larger scale, without contradicting the agile principles.

6.3 RQ3: Are the coordination methods unique for agile projects, or are they well known from other industries as well?

Communities of practice are not something unique made for agile projects. According to Wenger et al. (2002), “every organization and industry has its own history of practice-based communities, whether formally recognized or not”. However, Paasivaara and Lassenius (2014) shows that communities of practice was a key to the success of the agile transformation in an R&D organization within telecom company Ericsson, consisting of 400 employees.

When it comes to architectural methods, these are not something that only exist within agile methodology either. Actually, architecture was an important part of waterfall approaches. However, the way architecture are dealt with in agile projects are different. In the waterfall approach, architecture was often looked upon as a separate stage, finished before the development phase. In agile methods, architecture work are supposed to be more flexible, due to the agile principle of welcoming change, even late in the development (Beck et al., 2001).

6.4 RQ4: Is there an approximate linear correlation between number of teams (or number of team members), and how strict the coordination are?

Based on the scaled scrum framework Larman and Vodde (2013), there are a slightly more strict coordination when switching from framework-1 to framework-2. However, while in non-agile projects, a common way of dealing with coordination issues is to hire more managers. This is not the general case in agile methods, however there are added a new layer of area managers in framework-2.

7 References

Abrahamsson, P., Oza, N. & Siponen, M. T. 2010. Agile Software Development Methods: A Comparative Review. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Anderson, D. J. 2010. *Kanban: Successful Evolutionary Change for Technology Organizations*, Blue Hole Press.

Baskerville, R., Pries-Heje, J. & Madsen, S. 2010. From Exotic to Mainstream: A 10-year Odyssey from Internet Speed to Boundary Spanning with Scrum. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Beck, K. 1998. Extreme programming: A humanistic discipline of software development. *In:* Astesiano, E. (ed.) *Fundamental Approaches to Software Engineering.* Springer Berlin Heidelberg.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mallor, S., Shwaber, K., Sutherland, J. & Thomas, D. 2001. *The Agile Manifesto* [Online]. Available: http://agilemanifesto.org/ [Accessed April 16th 2015].

Bick, S., Scheerer, A., Spohrer, K., Kude, T. & Heinzl, A. Software Development in Multiteam Systems: A Longitudinal Study on the Effects of Structural Incongruences on Coordination Effectiveness. 9th Pre-ICIS International Research Workshop on Information Technology Project Management (IRWITPM 2014), 2014. 49.

Boehm, B., Lane, J. A., Koolmanojwong, S. & Turner, R. 2010. Architected Agile Solutions for Software-Reliant Systems. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Britto, R., Usman, M. & Mendes, E. 2014. Effort Estimation in Agile Global Software Development Context. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Chau, T. & Maurer, F. 2004. Tool Support for Inter-team Learning in Agile Software Organizations. *In:* Melnik, G. & Holz, H. (eds.) *Advances in Learning Software Organizations.* Springer Berlin Heidelberg.

Chow, T. & Cao, D.-B. 2008. A survey study of critical success factors in agile software projects. *Journal of Systems and Software,* 81**,** 961-971.

Coad, P., Lefebvre, E. & De Luca, J. 1999. *Java Modeling in Color with UML: Enterprise Components and Process*, Prentice Hall PTR.

Cockburn, A. 2004. *Crystal Clear: A Human-Powered Methodology for Small Teams*, Pearson Education.

Cockburn, A. & Highsmith, J. 2001. Agile Software Development: The People Factor. *Computer,* 34**,** 131-133.

Cockburn, A. & Williams, L. 2003. Agile software development: it's about feedback and change. *Computer,* 36**,** 0039-43.

Conboy, K. & Morgan, L. 2010. Future Research in Agile Systems Development: Applying Open Innovation Principles Within the Agile Organisation. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Dietrich, P. 2007. Strategies for coordination in complex multi-team development programs.

Dietrich, P., Kujala, J. & Artto, K. 2013. Inter-Team Coordination Patterns and Outcomes in Multi-Team Projects. *Project Management Journal,* 44**,** 6-19.

Dingsøyr, T., Dybå, T. & Moe, N. B. 2010a. Agile Software Development: An Introduction and Overview. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Dingsøyr, T., Dybå, T. & Moe, N. B. 2010b. *Agile Software Development: Current Research and Future Directions*, Springer Science & Business Media.

Dingsøyr, T. & Moe, N. B. 2014. Towards Principles of Large-Scale Agile Development. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. 2014. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation: XP 2014 International Workshops, Rome, Italy, May 26-30, 2014, Revised Selected Papers*, Springer International Publishing.

Dingsøyr, T., Nerur, S., Balijepally, V. G. & Moe, N. B. 2012. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software,* 85**,** 1213-1221.

Drury, M., Conboy, K. & Power, K. 2012. Obstacles to decision making in Agile software development teams. *Journal of Systems and Software,* 85**,** 1239-1254.

Dybå, T. & Dingsøyr, T. 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology,* 50**,** 833-859.

Dybå, T. & Dingsøyr, T. 2009. What Do We Know about Agile Software Development? *IEEE Software,* 26**,** 6-9.

Eckstein, J. 2014. Architecture in Large Scale Agile Development. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Edmonds, E. A. 1974. A process for the development of software for non-technical users as an adaptive system. *General Systems,* 19**,** 8.

Eklund, U., Holmström Olsson, H. & Strøm, N. J. 2014. Industrial Challenges of Scaling Agile in Mass-Produced Embedded Systems. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Faraj, S. & Sproull, L. 2000. Coordinating Expertise in Software Development Teams. *Management Science,* 46**,** 1554-1568.

Finne, H. 2014. Utfordringer og gode grep i store IKT-investeringer i offentlig sektor. SINTEF Teknologi of samfunn.

Fox, D., Sillito, J. & Maurer, F. Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry. Agile, 2008. AGILE '08. Conference, 4-8 Aug. 2008 2008. 63-72.

Gilb, T. 2007. Evo - Evolutionary Project Management.

Hellmann, T. D., Hosseini-Khayat, A. & Maurer, F. 2010. Agile Interaction Design and Test-Driven Development of User Interfaces – A Literature Review. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Highsmith, J. A. 1999. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House.

Iivari, J. & Iivari, N. 2010. Organizational Culture and the Deployment of Agile Methods: The Competing Values Model View. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Jacobson, I., Booch, G. & Rumbaugh, J. 1999. *The unified software development process*, Addison-Wesley Reading.

Kahkonen, T. Agile Methods for Large Organizations - Building Communities of Practice. Agile Development Conference, 2004, 22-26 June 2004 2004. 2-10.

Laanti, M. 2014. Characteristics and Principles of Scaled Agile. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Larman, C. & Basili, V. R. 2003. Iterative and Incremental Developments: A Brief History. *Computer,* 36**,** 47-56.

Larman, C. & Vodde, B. 2013. Scaling Agile Development. *CrossTalk***,** 9.

Lave, J. & Wenger, E. 1991. *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press.

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

Lui, K. M., Barnes, K. A. & Chan, K. C. C. 2010. Pair Programming: Issues and Challenges. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Martin, A., Biddle, R. & Noble, J. 2010. An Ideal Customer: A Grounded Theory of Requirements Elicitation, Communication and Acceptance on Agile Projects. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Nerur, S., Cannon, A., Balijepally, V. G. & Bond, P. 2010. Towards an Understanding of the Conceptual Underpinnings of Agile Development Methodologies. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Nord, R. L., Ozkaya, I. & Kruchten, P. 2014. Agile in Distress: Architecture to the Rescue. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Nyfjord, J., Bathallath, S. & Kjellin, H. 2014. Conventions for Coordinating Large Agile Projects. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Oxford English Dictionary *"scrimmage | scrummage, n."*, Oxford University Press.

Paasivaara, M. & Lassenius, C. 2014. Communities of practice in a large distributed agile software development organization – Case Ericsson. *Information and Software Technology,* 56**,** 1556-1577.

Paasivaara, M., Väättänen, O., Hallikainen, M. & Lassenius, C. 2014. Supporting a Large-Scale Lean and Agile Transformation by Defining Common Values. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. & Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Palmer, S. R. & Felsing, J. M. 2002. *A Practical Guide to Feature-driven Development*, Prentice Hall PTR.

Power, K. 2014. A Model for Understanding When Scaling Agile Is Appropriate in Large Organizations. *In:* Dingsøyr, T., Moe, N. B., Tonelli, R., Counsell, S., Gencel, C. &

Petersen, K. (eds.) *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.* Springer International Publishing.

Sharp, H. & Robinson, H. 2010. Three 'C's of Agile Practice: Collaboration, Co-ordination and Communication. *In:* Dingsøyr, T., Dybå, T. & Moe, N. B. (eds.) *Agile Software Development.* Springer Berlin Heidelberg.

Stapleton, J. 1997. *DSDM, Dynamic Systems Development Method: The Method in Practice*, Cambridge University Press.

Sutherland, J. 2001. Agile can scale: Inventing and reinventing scrum in five companies. *Cutter IT Journal,* 14**,** 5-11.

Sutherland, J. & Schwaber, K. 1995. Business Object Design and Implementation Workshop. *SIGPLAN OOPS Mess.,* 6**,** 170-175.

Sutherland, J. & Schwaber, K. 2010. *The Scrum Guide* [Online]. Available: http://www.scrumguides.org/ [Accessed May 2015].

Takeuchi, H. & Nonaka, I. 1986. The New New Product Development Game. *Harvard Business Review,* 64**,** 137-146.

Wenger, E. 1999. *Communities of Practice: Learning, Meaning, and Identity*, Cambridge University Press.

Wenger, E., McDermott, R. A. & Snyder, W. 2002. *Cultivating Communities of Practice: A Guide to Managing Knowledge*, Harvard Business School Press.

8 Appendices

8.1 Appendix 1: Interview guide

Written by Torgeir Dingsøyr

Intervjuguide

Introduksjon: Vi redegjør for bakgrunnen for prosjektet og hvordan vi skal anvende informasjonen vi får

- Hvorfor samtale, og hva blir data benyttet til
- Opptak av samtalen
- Anonymitet av person
- All deltakelse er frivillig (respondenten kan trekke seg før/under intervjuet)
- Varighet – ca 2 timer
- Prosjektslutt

Bakgrunnsinformasjon

Stilling / rolle - kan du beskrive din rolle i prosjektet

- oppgaver du hadde
- hva arbeidshverdagen besto i
- ansvarsområder
- tidslinje

Prosjektmandatet, og selve prosjektet

- beskrive mandatet
- prosjektstruktur
- roller og organisering – roller i de enkelte teamene
- planer, planleggingsgrad
- grad av detaljering – kontinuerlig planlegging
- Kontrakt og usikkerhet – hvordan ble dette håndtert – hvem var involvert og hvem ledet fra start, og underveis i prosjektet
- Formell/uformell ledelse

Prosjektsuksess

- Hva er prosjektsuksess for deg? Definer suksess (egne og prosjektets, samt organisasjonens)
  - – hvilke kriterier ligger til grunn'?
- Samarbeidsrelasjoner med kunde, medleverandør
  - ulike "regimer" (ønsker at disse beskrives)

    - hvilke ledelsesmessige grep/ prosesser/ beslutninger vil du trekke fram som sentrale elementer som disse personene bidro med for suksessen til prosjektet
- Beskriv tipping "points" – hendelser hvor ledelsen evt skiftet prosjektretning eller kurs (kan være en ytre eller indre hendelser, kan være mulighet eller trussel som oppsto eller ble oppdaget).

Metode

- hvordan ble smidig brukt i kobling mot "Method One"
- Perform-modellen – beskriv hva som ligger til grunn
- Hvilke mer universelle ledelsesmessige tiltak la man vekt på underveis i prosjektet – (daglig ledelse, ukentlig og eventuelt andre sentrale beslutningspunkt)

Overføringsverdi

- læringseffekten – rådgi andre-
    - hvilke tre faktorer ville du trukket frem ?