Research Project

Coordination Effectiveness in Large-scale Agile Software Development

December 11, 2014

Author:
Espen Andreassen

Advisor:
Torgeir Dingsøyr

NTNU – Trondheim
Norwegian University of
Science and Technology

Department of
Computer and Information Science

Abstract

Preface

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

I am now entering my last year on a master degree in computer science where I specialise in software, or more specifically, software systems. I was introduced to agile development methodologies through different subjects at the "Norwegian University of Science and Technology", NTNU, and also got hands-on experience working with Scrum in a subject called "TDT4290 - Customer Driven Project". This subject in particular sparked my interest in agile development methodologies and the new ways of handling work and project organisation. After a summer internship with EY (former known as Ernst&Young) I got more intrigued with how communication and coordination was handled in real life business and IT projects. Therefore, my previous experiences led to a motivation in exploring the combination of agile development and coordination.

1.2 Problem Description and Background

Since the introduction of agile development methodologies their usage have seen a steady growth. This has led to an increasing need for studies that reflect on the consequences and different aspects following the paradigm shift. One of these aspects is how coordination is handled [1–4]. At the International Conference on Agile Software Development (XP2013) "Inter-team coordination" was voted the number one burning topic in large-scale agile software development, with "Large project organization" coming in second [5]. In the latest years there has evidentially been an increase in companies and organisations performing development through agile development methodologies in large-scale projects [5–11], but the effects have not been well-documented [6, 12–16]. In my study this topic will be highlighted with the focus on coordination in large-scale agile projects. Theory, literature and models from the Software-field will be used and compared to other fields to see which changes and similarities the paradigm shift has brought forth (theories and literature from large-scale will be used where this is available).

Further, the Software-field especially has moved towards a higher degree of uncertainty and chaos, mainly because of the urge to be first-to-market and technology in constant change. This has produced an increasing need for flexibility in every stage of production and development [12,17,18]. The combination of the increasing need for flexibility, as well as the acknowledgement of effective coordination as an important part of organisations and their projects has led to the research question:

> How does coordination affect the level of effectiveness achieved in large-scale agile software development projects?

The purpose of the study and the planned master thesis will therefore be a combination of "To add to the body of knowledge", "To solve a problem", "To find the evidence to

inform practice", "To develop a greater understanding of people and their world" and "To contribute to other people's well-being" [19].

While research in small-scale agile software development is starting to get a good track record [6, 14], there is a clear gap in the research surrounding coordination in large-scale agile software development [5, 6, 12], and large-scale agile software development in general [13, 14]. Therefore, this literature study, as well as a planned master thesis, will contribute in filling parts of the gap. This will involve "An exploration of a topic, area or field", as well as "An in-depth study of a particular situation" in the case study planned for the master thesis [19].

As stated above, small-scale agile software development research is starting to get a good track record with successful findings. Because of these findings large organisations have been interested in adopting the benefits agile software development has shown over traditional development methods [1, 6–8]. The assumption that agile methodologies will deliver the same benefits when scaled to larger organisations and projects is therefore an interesting topic.

The combination of filling the gap and looking at the aforementioned assumption will be the pillars in the research outcomes.

1.3 Scope and Limitations

Because of the time constraints put on the research project it is obvious that some attention must be aimed towards the scope of the report and the limitations this implies. As mentioned in the previous section 1.2 large-scale agile projects, and agile projects in general, are growing in numbers. With this growth a lot of questions and interesting research problems arise. This particular research project only aims to cover the described research question: "How does coordination affect the level of effectiveness achieved in large-scale agile software development projects?".

Further, the research project does not aspire to introduce a brand new theory regarding the combination of large-scale, agile software development, coordination and effectiveness. The objective is to find and categorise research performed concerning the combination of these themes and look for common conclusions in their findings, as well as identifying and calling attention to clear gaps that need to be filled in the research field.

To give some insight and a clearer picture of the study, theory from agile software development, coordination and large-scale will be presented. Findings from a literature review will also be given on the combination of the aforementioned themes. It is important to note that the focus on coordination will primarily be on coordination across teams and not on coordination within these teams.

Lastly, the research study will not focus on frameworks and electronic tools suggested to support the large-scale agile processes. In this study the focus will rather be aimed towards empirical studies performed on the research area.

1.4 Report Outline

Chapter 2

Theory

Contents

2.1 Software Development Methodologies

The term software development methodologies has been around for quite some time now. These methodologies are frameworks for accomplishing a well-structured development process. In this section a brief introduction to the most prominent methodologies will be carried out. It will start with a quick look at the traditional software development, before ending with a presentation of the new and agile way of thinking. In the last section (on agile software development) the main focus will be on Scrum as this is the methodology found in most of the literature gathered from the literature review.

2.1.1 Traditional Software Development

Traditional software development methodologies have a distinct pattern. This pattern is sometimes called software development life cycle (SDLC) methodologies which is often found in system engineering. These "life cycles" are in contrast to the "iteration"-approach found in agile methodologies, such as Scrum. The most well-known of these traditional software development methodologies is Waterfall discussed further below.

Waterfall

The Waterfall methodology is one of the classic development models. It was first described in a paper by W. W. Royce in 1970 [20]. The model was not yet named in this paper, which it received later mostly due to its iconic structure (as shown in figure 2.1).

In the aforementioned paper, it is suggested that all software development models tend to go through two distinct phases: Analysis and Coding. The author argues that it is not possible to write a software project without having a somewhat deep understanding of the underlying problems that it needs to solve. Therefore an analysis phase will always be required in advance of writing the program itself. However, he also mentions that such a simple model is only suitable for programs that are completed in a matter of days. Larger software projects require an extended number of steps.

For larger projects, the following steps are suggested:

1. System and Software Requirements: The customer is involved with the specification of the scope and requirements of the system. The resulting documentation serves as a foundation to the next stages of development.

2. Analysis and Program Design: The requirements produced in the previous stage are used to create a system plan and various design documents.

3. Coding and Testing: The actual implementation of the project. This also involves continuously testing on various levels (for example unit and integration).

4. Operation and Maintenance: Once the project has been completed, it has to be maintained during its usage. In addition to improving the program in various ways,

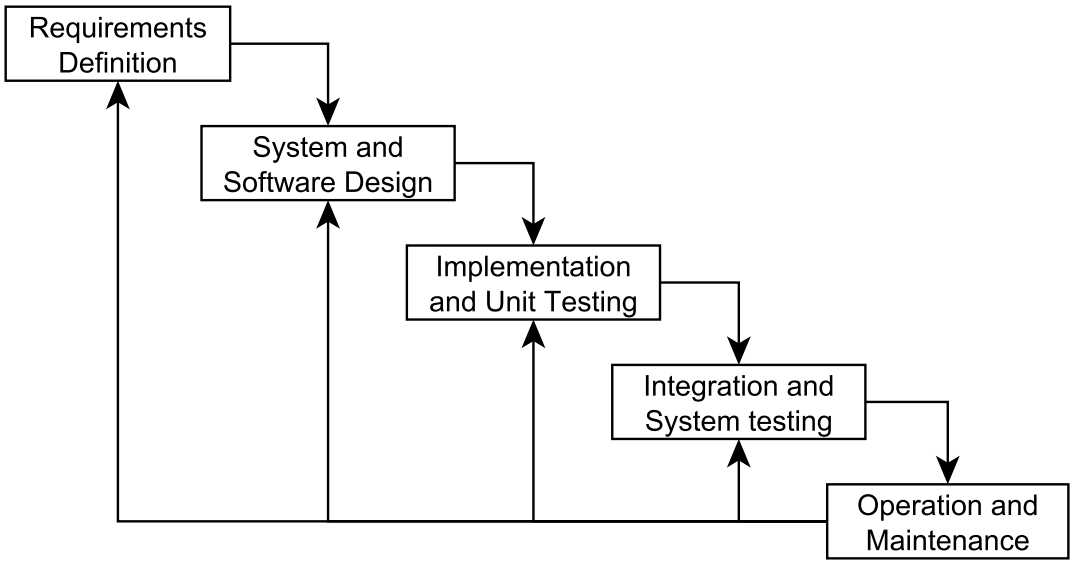

Figure 2.1: The Waterfall model.

this may also involve the inclusion of extra features if the customer so desires. These features can in themselves use the Waterfall model.

The model initially suggested by W. W. Royce discusses a linear model in which each of the aforementioned stages are used as distinct steps in the development process. Each stage is required to be completed before the next is started. This may be a sound premise in theory, but as suggested in the paper it is likely to fail in practice. The argument used is that often during development, unforeseen problems in the design are encountered. The linear model does not allow for a return to a previous stage in development. Hence, it does not allow for changes in the design that could potentially resolve such problems.

Therefore, an alternative model is suggested that allows for the process to return to earlier stages if necessary. This may not be an ideal solution either, but it does allow for encountered problems to be addressed during development.

2.1.2 Agile Software Development

As can be seen from the ending of the Waterfall-section there were doubts about its applicability already at an early stage. With the advancement of business needs and customer involvement something had to change. This opened the door for the introduction of a new software development methodology, namely agile software development. This new way of thinking tries to deal with collaboration in a way that promotes adaptive planning, early delivery and continuous improvement, making the development phase faster and more flexible regarding changes [21].

Scrum

In this section an introduction to one of the most popular agile software development methodologies will be carried out based mainly on Abrahamsson, Salo, Ronkainen and Warsta's publication on agile methods [21], the so-called Scrum. In VersionOne's "7th Annual State of Agile Development Survey" Scrum or Scrum variants had a quoted 72

Scrum is an iterative and incremental software development model (as shown in figure 2.2). It has come forth from the realisation that development methods that were common at the time of its introduction worked well in theory but did not in practice. These methods, Waterfall included, were designed to provide a structured and well-defined development process [22].

The agile software development processes, like Scrum, are part of a recent approach to software development. The idea with Scrum in particular is to divide the development into short periods called "sprints". This is done to focus effort for a limited time on short-term goals. Iterating over these goals allows the process to adapt the development plan based on progress but also to address any design problems that arise.

In short, the team concentrates on isolated parts, and through this prioritises on the most important tasks of the project first. The time span of a sprint is typically between one and four weeks long.

In order to implement the requirements step by step and in an orderly fashion, a repository is kept containing the features that have yet to be implemented. This repository is called the "product backlog". During development, the requirements could change over time. Therefore the product backlog is not static; it changes to the needs of the project with new topics being added, and obsolete ones being removed. The items from the backlog that a team works on during a sprint is called the "sprint backlog".

Meetings are also a key part of Scrum. There are several different types of meetings: sprint planning meeting, daily scrum meeting, backlog refinement, end of cycle and Scrum-of-Scrums. The sprint planning meeting is held at the beginning of each sprint cycle. Here the focus is on what work is to be done, and the sprint backlog for the coming sprint cycle is set. The daily scrum meeting, also called the daily stand-up, is a daily encounter (15 minutes) where each member of the project team answer these three questions:

1. What have you done since yesterday?
2. What are you planning to do today?
3. Are there any impediments in your way?

Further, there is the backlog refinement, also called "grooming". This is where tasks are created, large tasks are decomposed into smaller ones, tasks are prioritised, and the existing tasks are sized in the product backlog. Backlog refinement is often split into two meetings. In the first meeting the product owner and other stakeholders create and refine stories in the backlog. In the second meeting the project team sizes the tasks in

the backlog to make them ready for the next sprint. Planning poker is an example of how this can be carried out.

The last listed meeting occurs at the end of each cycle, and is therefore called end of cycle (meeting). This is actually two meetings: a sprint review meeting and a sprint retrospective. At the sprint review meeting the work that is completed and yet to be finished is reviewed. The completed work is also presented for the stakeholders, often called "the demo". At the sprint retrospective all members reflect on the past sprint. Two main questions are answered:

1. What went well during the sprint?
2. What could be improved in the next sprint?

The Scrum team usually consists of five to nine members. It is important to note that Scrum teams do not use traditional roles such as programmer, tester, designer or architect. Instead the main goal for the Scrum team is to collectively complete the tasks within the sprint.

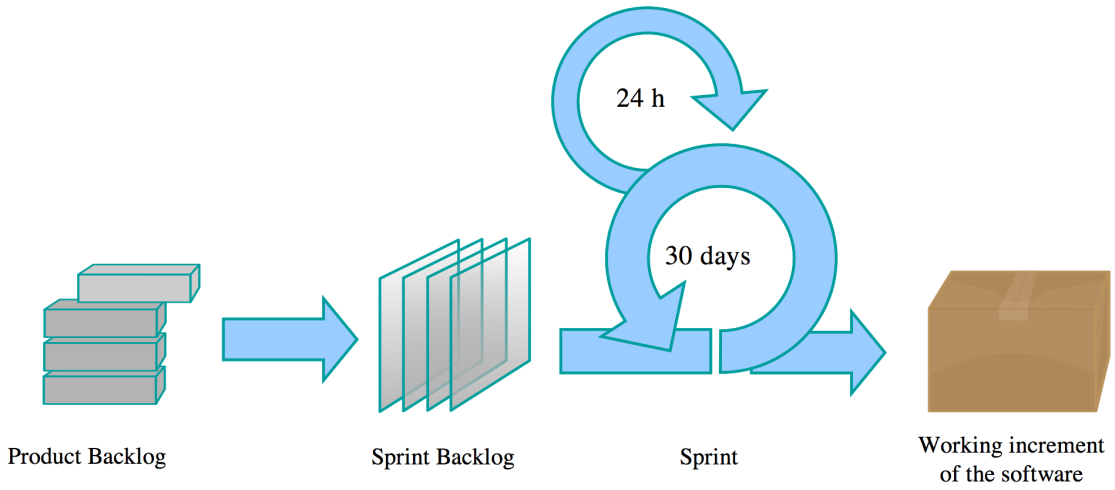

Figure 2.2: The Scrum cycle.

To end the section, as well as making a natural shift towards the next topic (Coordination), a look at Scrum-of-Scrums is carried out. It is a natural shift because Scrum-of-Scrums are used as the coordination mechanism across teams in the Scrum methodology. It works as the daily scrums (though usually implemented on a weekly basis because of time constraints and the complexity to find common times for all teams), but with one member assigned from each Scrum team to report completions, next steps and impediments for their respective teams. It is important that these impediments focus on the challenges that may impact coordination across teams and might limit other teams' work. The Scrum-of-Scrums will have their own backlog aiming to improve the cross-team coordination [23]. Below the suggested questions for the SoS meetings are listed [24]:

1. What did your team do since the previous meeting that is relevant to some other team?
2. What will your team do by the next meeting that is relevant to other teams?
3. What obstacles does your team have that affect other teams or require help from them?
4. Are you about to put something in another team's way?

2.2 Coordination

2.2.1 Malone and Crowston's Coordination Theory

One of the most well-known papers on coordination theory was published by Malone and Crowston in 1990 and further redefined in 1994 (the focus will be on this paper) [25]. Their study spans different fields and can therefore be seen as an interdisciplinary coordination study. They list an extensive amount of different definitions of coordination, and through these proposed definitions and their own work come up with a rather simple definition:

> Coordination is managing dependencies between activities.

These dependencies can occur when some task has to be postponed or extended because of its connection to another task, resource or unit. Their theory is based on a combination of coordination from several different disciplines such as computer science, organization theory, operations research, economics, linguistics, and psychology. They state that coordination consists of one or more coordination mechanisms, and that each of these address one or more dependencies.

While Strode et al. acknowledges their coordination theory as very useful for identifying these so-called dependencies, categorising them, and identifying coordination mechanisms in a situation, they conclude that it is only a theory for analysis and not intended to be used for prediction. Despite this being true, and the coordination theory not being suitable for predicting outcomes such as coordination effectiveness, their theory adds important information for better understanding of how activities or artefacts support coordination in organisational settings [26].

2.2.2 Strode's Theoretical Model of Coordination

Strode et al. performed a multi-case study on three different co-located agile projects in 2012 [26]. From these projects the findings led to a theoretical model of coordination

that will be outlined in this section. It is important to note that these projects were not large-scale, but the model will nonetheless be used to compare if there are similarities from the model proposed by Strode et al., and the findings from the literature review on large-scale agile project coordination. This will be performed in later sections.

From these case studies three main components for the theoretical model were extracted: Synchronisation, Structure and Boundary Spanning. These components combine to what is called the "Coordination Strategy". Coordination strategy is in this context a group of coordination mechanisms that manage dependencies in a situation. The theoretical model of coordination can be seen in figure 2.3. Below the three main components will be explained in more detail:

Synchronisation

Synchronisation in this context consists of synchronisation activities and synchronisation artefacts produced and used during these activities. Synchronisation activities are activities performed by all team members simultaneously. They contribute to a common understanding of the task, process, and or expertise of other team members. Synchronisation artefacts on the other hand are artefacts that are generated during synchronisation activities. These artefacts may be visible for the entire team or largely invisible but available. The artefacts can take a physical or virtual form, and are temporary or permanent.

Structure

Structure in this model is the arrangement of, and relations between, the parts of something complex. It consists of three categories: proximity, availability and substitutability. Proximity is the physical closeness of other (individual) team members. Availability means that other team members are accessible for requests or information. Lastly, substitutability has to do with the team members ability to perform others' work to maintain time schedules.

Boundary Spanning

The last component of the coordination strategy is boundary spanning. Boundary spanning has to do with the interaction with other organisations or other business units that are not involved in the project. It consists of three aspects: boundary spanning activities, boundary spanning artefacts and a coordinator role. Boundary spanning activities are activities performed to achieve help from some unit or organisation not involved in the project. The boundary spanning artefacts are artefacts produced to enable this external coordination. These artefacts have the same characteristics as synchronisation artefacts. Lastly, the coordinator role is a role taken by someone within the project team. His or her role is to support interaction to outside personnel to extract resources or information needed in the project at hand.

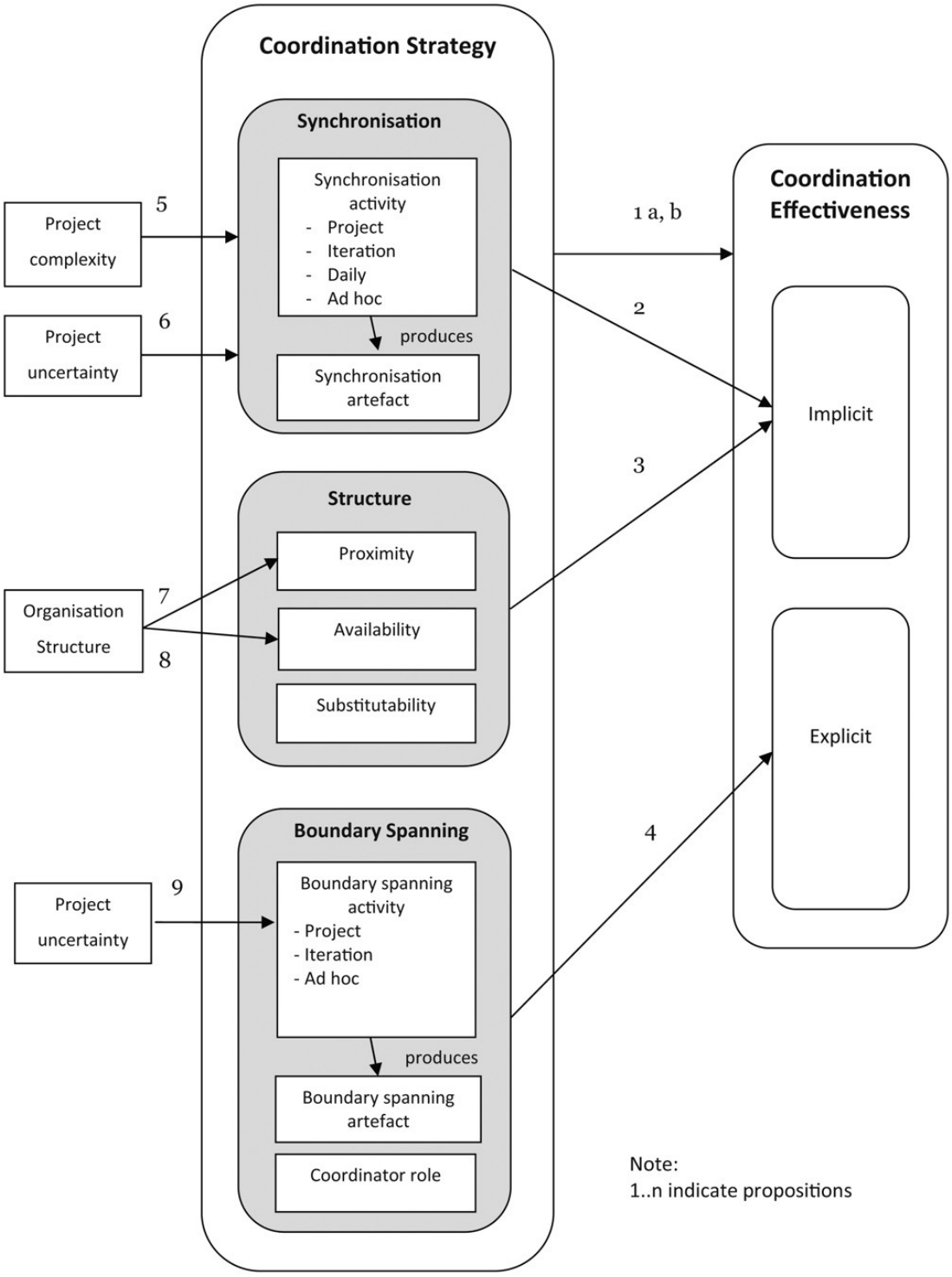

Figure 2.3: A theory of coordination in agile software development projects.

Coordination Effectiveness

There is another important part of the theoretical model of coordination, namely the coordination effectiveness concept. This concept will be further explained in section 2.4 that takes a look at coordination effectiveness.

2.2.3 Coordination in Large-scale

2.3 Large-scale

Having looked at coordination in large-scale, what is actually so-called "large-scale"? This was a topic brought up at a workshop regarding research challenges in large-scale agile software development where opinions ranged by some margin. Some suggestions were project duration, project cost, people involved, number of remote sites and number of teams [5]. This issue was further analysed by Dingsøyr, Fægri and Itkonen trying to work out a taxonomy of scale for agile software development. Their results are summarised in table 2.1 where the taxonomy of scale is based on the amount of teams involved in the development project [15].

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

Table 2.1: A taxonomy of scale of agile software development projects.

Others have also discussed the problems regarding large-scale. For example Schnitter and Mackert discuss the scaling of Scrum at SAP AG and concludes that in their case the maximum involved development employees that may be organised with regards to agile project management is 130 (This number sums up developers in 7 teams (max. 70 people), the product team (max. 16), development infrastructure responsibles (about 10), quality assurance and testers (about 25), general management (about 10)) [27].

Another example is taken from Nord et al. defining large-scale by scope of the system, team size, and project duration. They say that the size of the development team must be more than 18 people and distributed into a few teams [28].

So the definition of a "large-scale agile project" used in this research will be:

> An agile project must consist of a minimum amount of two teams coordinating across the teams to be categorised as large-scale.

2.4 Efficiency, Effectiveness and Productivity in Coordination

There has been released a good amount of papers regarding effectiveness, productivity and efficiency in project literature. Unfortunately research in this area that focuses on large-scale is scarce. Therefore, the work highlighted in this section will mainly be extracted from small-scale studies. To start the section of a closer look at the aforementioned study by Strode et al. will be performed, before a summary of some different field studies on the matter will be carried out.

2.4.1 Strode's Coordination Effectiveness

Part of the theoretical model of coordination by Strode et al. seen in figure 2.3 is the so-called "coordination effectiveness". This concept was developed by Strode et al. in 2011 having used the same three agile projects discussed earlier, as well as a non-agile software development project as a base [29]. Coordination effectiveness is defined as the outcome of a particular coordination strategy. Coordination effectiveness is split into two components: an implicit and an explicit.

The implicit part is concerned with coordination that occurs without explicit speech or message passing, this happens within work groups. It has five components: "Know why", "Know what is going on and when", "Know what to do and when", "Know who is doing what", and "know who knows what". These aspects are pretty self-explanatory.

The explicit component on the other hand is concerned with the physical aspects of the project. It states that the objects involved in the project have to be in the correct place, at the correct time and in a state of readiness for use. A summary of the combination of explicit and implicit coordination effectiveness is provided in figure 2.4.

To end this subsection a definition of coordination effectiveness from Strode et al. is provided:

> Coordination effectiveness is a state of coordination wherein the entire agile software development team has a comprehensive understanding of the project goal, the project priorities, what is going on and when, what they as individuals need to do and when, who is doing what, and how each individuals work fits in with other team members work. In addition, every object (thing or resource) needed to meet a project goal is in the correct place or location at the correct time and in a state of readiness for use from the perspective of each individual involved in the project [29].

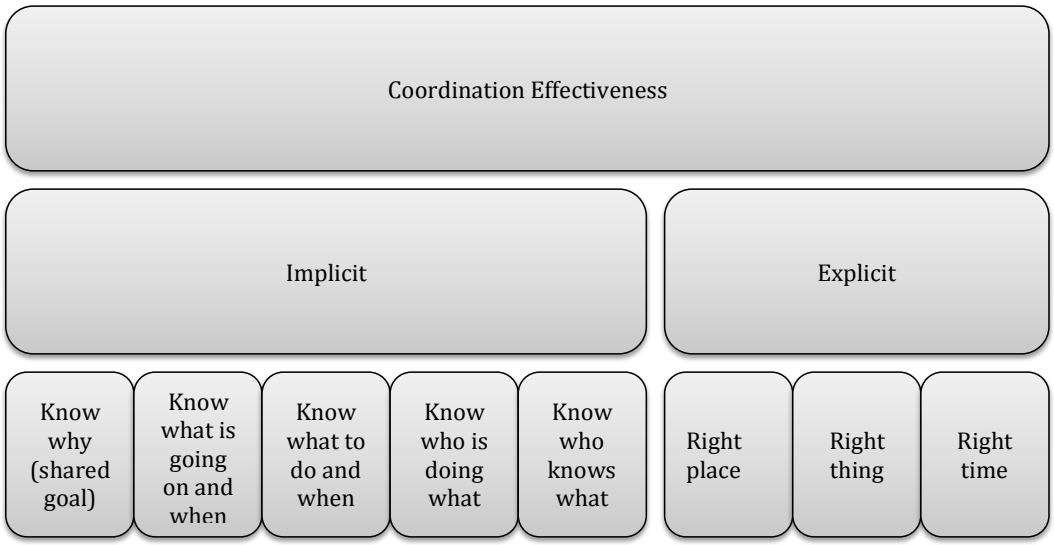

Figure 2.4: Components of coordination effectiveness from Strode et al. (2011).

2.4.2 Some Studies on the Field

Team Effectiveness 1997-2007: A Review of Recent Advancements and a Glimpse Into the Future

Interpretative Case Studies on Agile Team Productivity and Management

Melo et al. performed a multi-case study on three large Brazilian IT companies that were using agile methods in their projects [30]. The objective of the research was to provide a better understanding of which factors that had an impact on agile team productivity. To document teamwork effectiveness they used the well-known theoretical model "Input-Process-Outcome" (IPO). Their input factors were "Individual and Group characteristics", "Stage of team development", "Nature of task", "Organizational context" and "Supervisory

behaviors". One process-category was identified: "Group processes". Lastly they identified two outcome-groups, namely "Agile team productivity" and "Attitudinal and Behavioral". All of these are summarised constituting the conceptual framework for their agile team productivity in figure 2.5.

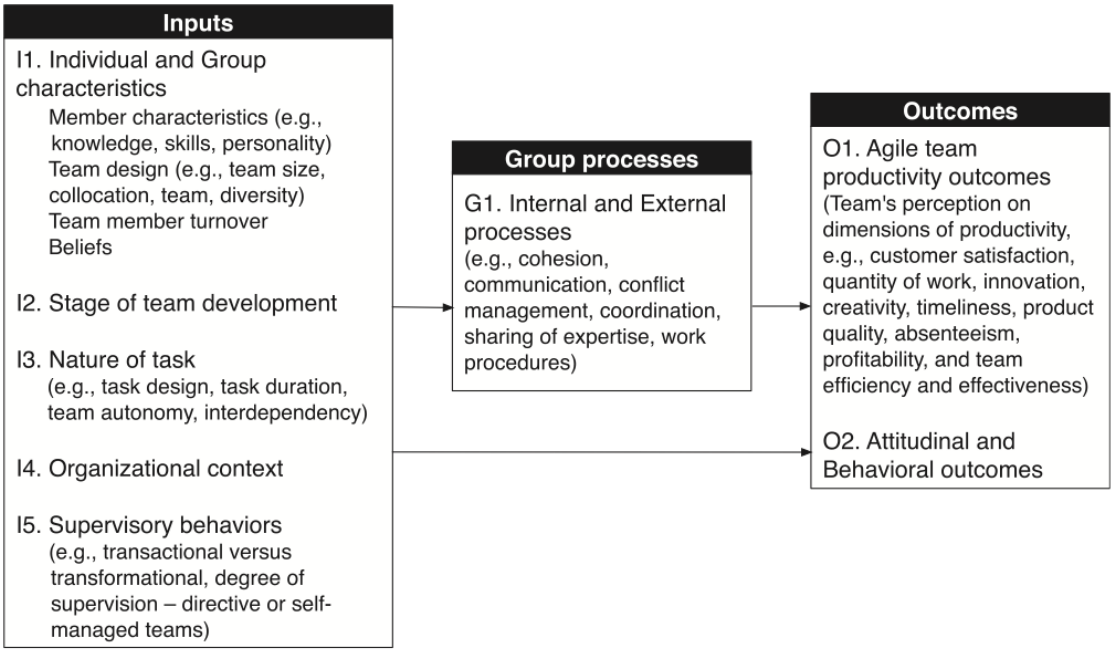

Figure 2.5: Agile team productivity conceptual framework.

After collecting the data from their multi-case study they mapped the results in a thematic map on agile productivity factors. These findings showed three main groups of team management and their impact on productivity. For this study it is the "Inter-team coordination" and "Team design choices" that are interesting because they have an impact on coordination in a larger degree, meaning "Team member turnover" is left out.

In "Team design choices" four roots of impact were identified: "Team size", "Team members skills", "Team collocation" and "Team members allocation". Out of these team collocation and team size seem to effect coordination effectiveness the most. Their findings showed that smaller teams led to better communication and alignment, while collocation had a positive influence on team productivity as it helped overcome invisible barriers between teams in a hierarchical company.

For "Inter-team coordination" two roots were identified: "Lack of commitment among teams" and "Inappropriate coordination rules among teams". One of the main reasons for negative impact was identified as external dependencies because projects often were left waiting for results of entities outside the project team. So a problem in inter-team coordination was misalignment, hence, synchronisation is an important factor.

Dispersion, Coordination and Performance in Global Software Teams: A Systematic Review

Anh et al. performed a systematic literature review (SLR) to collect relevant studies on dispersion, coordination and performance in global software development (GSD), and highlighted the findings of impact factors in a thematic mapping [31]. It is important to note that the findings are not from agile software development, but they are still interesting because of the global dispersion aspect in the literature used. The results are briefly summarised (degree of temporal dispersion is not included as the findings were inconclusive) in table 2.2:

Type	**Impact on team performance**
Presence of geographical dispersion	Negative (work takes longer time, less effective communication and coordination)
Number of sites/Team size	Negative (complicates coordination and hampers communication)

Table 2.2: Impact of geographical dispersion on performance.

Team Performance in Agile Development Teams: Findings from 18 Focus Groups

Dingsøyr and Lindsjørn carried out a focus group study looking at which factors that the agile software practitioners in the research perceived as influential on effective teamwork [32].

Summary

The findings from the different studies are summarised in table 2.3.

Type	**Impact**
Misalignment	Negative
Synchronisation	Positive
Team collocation	Positive
Presence of geographical dispersion	Negative (work takes longer time, less effective communication and coordination)
Number of sites/Team size	Negative (complicates coordination and hampers communication)

Table 2.3: Summary of impacts identified in the studies.

Chapter 3

Method

Contents

3.1 Snowball Sampling and General Accumulation

Snowball sampling is a statistical term that reflects how new studies are selected through already chosen studies (based on their similarities) [33]. This was done in two ways in the research. In table 3.1 a list of databases used for a literature review are summarised. Some of these databases provided snowball sampling in the way of suggesting similar articles when a specific publication is selected from a search. This is the first way of snowballing used. The second way was through using references lists in selected articles and publications. This extraction lead to a lot of well-written and recognised papers.

Articles were also accumulated through a supervisor and fellow research students. It is important to note that all the articles were inspected in the same manner as the publications found from the literature review to make sure their relevance and rigour were appropriate.

3.2 Literature Review

For this study a structured systematic literature review was chosen as the information gathering method. The normal procedure of such a review is outlined in L1 below. It is important to note that because of the low amount of studies surrounding the research question the level of acceptance had to be lowered by a small margin (step 3 in L1).

L1 - The steps of a systematic review [34]:

1. Framing questions for a review.
2. Identifying relevant work.
3. Assessing the quality of studies.
4. Summarizing the evidence.
5. Interpreting the findings.

Some of the benefits and objectives of a literature review are summarised in L2 below.

L2 - Objectives of a literature review [19]:

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

As explained in section 3.1 above a set of articles and publications were provided by the supervisor to give an overview on the field and agile software development in general. This made it easier to classify which studies to look for and how to evaluate their relevance and rigour. The databases used in the literature review are summarised in table 3.1. When searching in these databases concepts and keywords were combined to match the research question as well as possible. These concepts and keywords are outlined in table 3.2. It is important to note that the last concept was an additional search word used because a lot of research seemed to either have focused on a co-located or a distributed manner.

Name	**Impact**
ISI Web of Science	apps.webofknowledge.com
ACM Digital Library	dl.acm.org
Science Direct (Elsevier)	sciencedirect.com
Google Scholar	scholar.google.com

Table 3.1: Databases used in the literature review.

Concept	**Keywords**
Coordination	Communication, collaboration
Agile	Scrum, XP, Crystal, Lean, Extreme Programming, Xtreme Programming
Large-scale	Global, multisystem, distributed, international
Effectiveness	Efficiency, productivity, performance
Location (Additional search words)	Co-located, collocated, colocated, co located, distributed, dispersed

Table 3.2: Search words used in the literature review.

The literature review provided an extensive amount of findings, unfortunately a lot of the publications were focusing on small-scale development. Therefore, a selection process had to be carried out. Here all abstracts of the collected literature were read and publications with the highest relevance were chosen. The articles that were still left after this selection process were then read thoroughly where some were discarded to give an appropriate amount of publications. This process was important because of the time constraints specified on the study, and to obtain relevant and rigorous literature to insure a robust study.

Chapter 4

Results

Contents

4.1 Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work?

As mentioned in section 2.2 Scrum-of-Scrums (SoS) are considered as the method for handling inter-team coordination in large-scale Scrum projects. In this article by Paasivaara et al. implementation of these so-called SoSs are looked at in a multi-case study of two globally distributed large-scale projects to investigate their success [6]. The projects were from two different telecom companies. Some information about the projects and data collection is summarised in table 4.1.

	Case A	**Case B**
Product	Telecommunications, started from scratch	Telecommunications, 10-years old
Process	Scrum	Incremental change from Waterfall to Scrum
Scrum experience	2,5 years	1,5 years
Sites and # of teams	Finland (10 dev. teams), India (6 dev. teams), Germany (2 test teams), Greece (2 test teams)	Finland (18 teams), Hungary (7 teams)
Interviews	19 (Finland 16, Greece 3)	39 (Finland 28, Hungary 11)
Roles (The sum exceeds the total number of interviews, as some line managers had double roles, e.g. also worked as Scrum Masters)	Managers (4), Agile coach (1), Scrum Master (1), Developers (5), Testers (2), Line managers (2), Area Product Owners (4), Architects (1)	Managers (6), Agile coach (1), Scrum Masters (6), Team members (13), Line managers (3), Product/Proxy product owners (7), Technical management / architecture (5)
Interview length	Managers, coach: 1.5-3h, others: 1-1.5h	Managers, coach: 2-3h, others 1-2h

Table 4.1: Case projects and data collection.

4.1.1 Findings from Project Cases

In Case A the SoS meetings were split into two separate meetings: a Finnish SoS and a Global SoS. The Finnish SoS involved one member from each of the ten development teams located in Finland, as well as the Finnish project manager. In the Global SoS the Finnish project manager led the meetings with one representative from each of the six global development teams and four global test teams present. Initially all questions listed in section 2.2 on Scrum-of-Scrums were answered, but this was later changed because

the information gathered from these questions were not perceived as important by all participants. This led to meetings where only impediments or planned impediments for other teams were discussed. Hence, a lot of teams reported to have no problems in their work, which often was not the case, but they felt the other teams did not need to know this information. Managers from Case A confessed that the inter-team communication in the different SoS meetings did not work properly.

In Case B similar issues were faced. Initially the SoS meetings were held only on a global scale, and representatives had problems identifying what to report (as in Case A), and often ended up reporting that there was nothing to share. Because of this a new type of SoS meetings were introduced, namely Feature SoS meetings. In these meetings only teams working on the same features and functionality were present, and all interviewees found the Feature SoSs to be useful. The global SoS meetings were still present, but renamed to Grande SoS meetings. These were still perceived as problematic as attendants found them as uncertain and unnecessary as before.

4.1.2 Conclusion

As can be seen from above it is obvious that the findings show problems with the growing number of participants, especially when these participants have separate interests and concerns. In Case B this was backed up by the success of the smaller Feature SoS meetings focusing on specific functionality. Both case projects also reported a need for project-wide inter-team synchronisation, but had not found any satisfactory solutions.

4.2 Communities of Practice in a Large Distributed Agile Software Development Organization – Case Ericsson

Paasivaara et al. extends the previously discussed study with a deeper analysis of Case B (will be referred to as Ericsson from now on). In the previous study the project consisted of 25 Scrum teams located at two remote sites (Finland and Hungary), this was later expanded. In this case study the Ericsson project involves 40 Scrum teams from the three global sites (Finland, Hungary and the US). The case extends the previous study by looking at a shift from the use of SoS meetings to Community of Practices (CoPs) and will be further outlined below [35].

4.2.1 Community of Practices Overview

The CoPs were introduced at the same time as the company made a shift from a traditional software development methodology to an agile approach. Ericsson took use of several different CoPs, the most prominent ones are summarised in table 4.2 with their key findings.

	Coaching CoP	**Feature CoP**	**Developer CoP**	**End-to-end CoP**
Predecessor	Scrum master CoP	SoS and system CoPs	Previous developers' CoP	Program weekly
Predecessor's role	Sharing experiences in applying Scrum practices	SoS: inter-team coordination, Syst.CoP: feature design	Design rules etc.	High-level R&D progress monitoring and way of working
Participants	Scrum masters/coaches	Representatives of cross-functional teams	Developers	Person wanting to make a difference: managers, product owners, coaches, team members
Current role	Improving the whole organization and its competences	Supporting coordination and design between teams developing a common feature	Software craftsmanship, unifying tools and technologies over product areas	Improve the way of working and optimize the end-to-end flow
Location and distribution	Cross-site (Finland/Hungary), and site-specific CoPs	According to the distribution of the product areas, mostly cross-site	Finland, Hungary	Finland, Hungary, US
Rhythm	Weekly	Feature coordination CoPs weekly, Feature design CoPs on a need basis	Bi-weekly	Weekly
Challenges faced	Lack of common activities and goals after day-to-day problems were solved	Organization-wide SoS did not work: several trials to find a solution to inter-team issues	The first trial ceased to exist due to lack of CoP culture	Involving the US site due to time-zone difference
Successes	Promising new start with goal to improve the whole organization and its competences	After many trials a functioning solution for inter-team coordination and design	Promising start-up after the death, with a passionate leader, interesting topics and goals	Broad representation of organizational levels and sites facilitates decision making and moving things forward

Table 4.2: Examples of CoPs at Ericsson.

As both the Coaching CoPs and Developer CoPs mainly focus on knowledge sharing these are not further looked into. The interesting CoPs from this study are the Feature CoPs and End-to-end CoPs because of their impact on coordination.

Feature CoP

As discussed in section 4.1.1 Feature SoS meetings evolved from the traditional SoS meetings after the troubles with reporting in global SoS meetings. These Feature SoS meetings were held on a weekly basis with approximately 3-8 teams involved in each meeting. Overall there seemed to be a content with these meetings as illustrated in the quotation below from a product owner at the Hungarian site:

> Feature SoS meetings are pretty good, because people participating in them work on the same things, talk "the same language" and have a common goal.

In parallel with these meetings Ericsson held organization-wide SoS meetings, but these perished. At the same time Feature SoSs were re-branded because of the negative connotations with the word "SoS". This led to the introduction of the so-called "Feature CoPs". New Feature CoPs were also introduced at the same time, e.g., System CoP.

These Feature SoS/CoP meetings grew further evolving to what was called the "Feature coordination CoPs" in the study. These meetings had a timeslot of one hour where about half of that was used at traditional SoS style coordination discussion, and the rest was aimed towards discussing general topics like testing, continuous integration and other improvements. They also witnessed the need to separate the most technical challenges into another meeting which evolved into Feature Design CoP meetings. These seemed to give additional benefits and were seen as well-functioning. As can be seen from the quotation below from one of the developers at the Finnish site, the Feature Design CoPs were self-organising, which is in line with the agile methodologies:

> These meetings (Feature Design CoPs) are invited by individuals, it is not Product Owner driven. When somebody feels that we need to do something, he takes the responsibility and invites a meeting. And luckily, there has been interest, and the people that have been needed have participated, and also those that have been interested to learn more have participated. Thus, it is not only the best gurus that discuss together. Instead, it is an open meeting that everybody who wants may participate in. And the invitation is send to everybody. I think it is very important, because if there will be cliques, bad things will happen.

End-to-end CoP

End-to-end CoPs were introduced to improve the organisation as a whole and to optimise product development flow throughout the whole organisation (removing bottlenecks along the way). The CoP meetings were organisation-wide, even attracting some of the US team members at times. The feedback surrounding the End-to-end CoP meetings had been very good. A quotation from one of the product owners at the Finnish site is included to give an overview of the meetings' contribution:

> There, we concentrate on what are the challenges, what we should do to be more efficient as an organization. If we don't have enough test environments, or certain knowledge, or the way-of-working is bad, or collaboration is not working. From a very broad scale you can bring in topics and there is the head coach and often other managers, people who can easily take things forward.

4.2.2 Characteristics of Successful CoPs

From the interviews in the study some characteristics for successful CoPs were extracted. All of them are summarised in figure 4.1. A few of these will be briefly described: "Interesting topic and concrete benefits", "Open community" and "Cross-site participation when needed".

It is important that CoPs have interesting topics so the participants will feel that they benefit from the meetings. This was especially the case for the Feature CoP meetings where participants had a clear interest in the aspects discussed. For the CoP culture

to work it is paramount that the community is open in the sense that they are self-organising. Forced CoPs or meetings that do not interest the participants will not work well. Lastly, it is important that cross-site participation is performed when needed in a globally distributed project. To keep synchronisation and coordination at a high level this is necessary. This was problematic in the initial organisation-wide SoS meetings, but was deemed successful in the End-to-end CoP meetings.

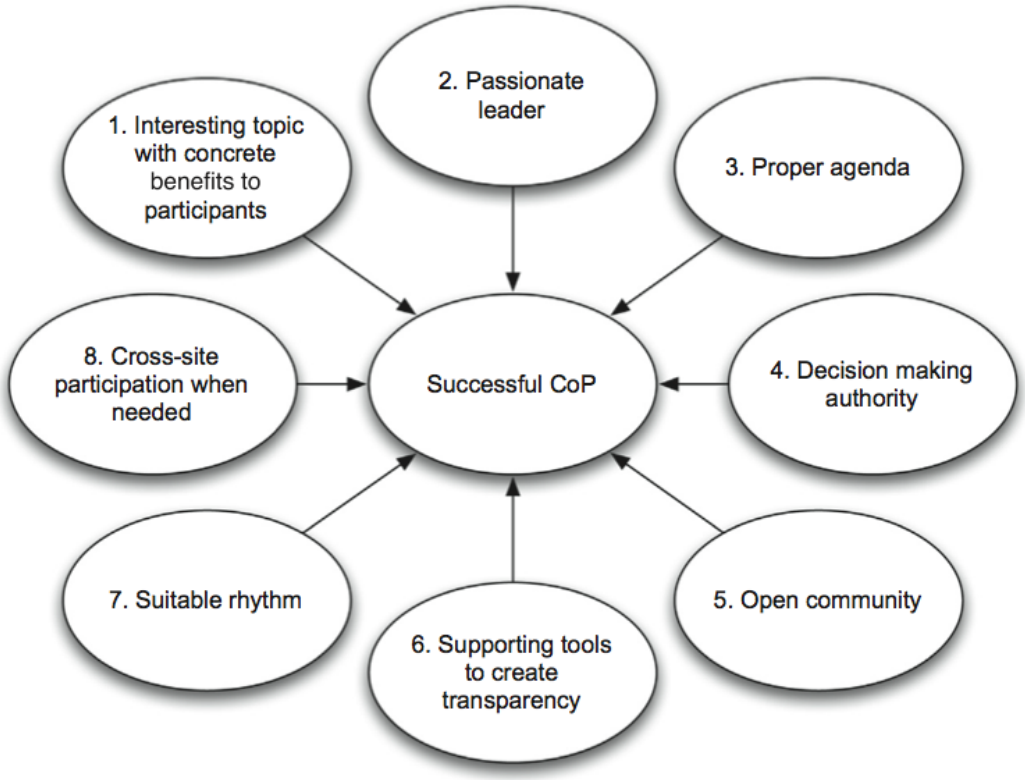

Figure 4.1: Characteristics of successful CoPs.

4.2.3 Purpose of CoPs

There were four purposes of the CoPs classified from the study. These are summarised in table 4.3 with an example of each. The focus here will mainly be on the coordination purpose with a quick look at the organisational development.

The aforementioned Feature Coordination CoPs was the main form of inter-team coordination achieved through CoPs. Here several cross-functional teams working on similar functionality or working inside one product area meet and coordinate their work. These types of CoPs were formed when the organisation-wide Scrum-of-Scrum meetings did not deliver expected benefits. In Scrum literature the SoS meetings are seen as a reporting meeting, but here these meetings evolved into discussion meetings focusing on coordination issues, namely the Feature Coordination CoP meetings. The organisational development purpose can indirectly also be seen as a coordination purpose. This has to do with the aim to improve and optimise the product development at an organisation-

wide level, which should lead to synchronisation improvements across teams. An example from the study was the End-to-end CoPs.

Purpose	**Example**
Knowledge sharing and learning	Role-based CoPs
Coordination	Feature coordination CoP
Technical work	System CoPs
Organisational development	End-to-end CoP

Table 4.3: CoP purposes at Ericsson.

4.2.4 Findings and Conclusion

The case study shows how CoPs can be used as a central mechanism for success when introducing a shift from a traditional software development methodology to a large-scale agile approach. CoPs were a central part in knowledge sharing, inter-team coordination and communication, technical work and organisational development. The characteristics for how Ericsson achieved successful CoPs are summarised in figure 4.1, while table 4.3 illustrates the purpose behind the CoPs. Regarding coordination it is important to note how CoPs gradually replaced the not so successful Scrum-of-Scrum meetings as Ericsson's inter-team coordination method.

Below practical implications are listed for the organisational and management level in the first list, and for the practitioners in the organisation in the second list.

Practical implications for the organisational and management level:

1. CoPs can support a Lean and Agile transformation. The study suggests that CoPs can be used as an effective mechanism for the transformation from a traditional software development methodology to an agile approach.

2. CoPs can support scaling agile to a large and distributed organisation. As basic Scrum does not support large-scale cross-team coordination this is an important finding.

3. Building a CoP-friendly corporate culture is important. For the CoPs to succeed this is crucial aspect.

Practical implications for the practitioners in the organisation:

1. Participate in CoPs and create a new one when needed. CoPs should be self-organising meaning that they are created and disbanded on a need basis.
2. Use CoPs to learn and further your career. CoPs are a good way of keeping things synchronised, as well as broadening the practitioners knowledge.
3. Influence the organisation via CoPs. Seeing decisions can be made within CoPs it is important that organisational members attend, and by that influence the direction of the organisation.

4.3 Towards a Governance Framework for Chains of Scrum Teams

Vlietland et al. takes a further look at the increasing adoption of agile methodologies in large companies. Their focus is on how the traditional chain of production is handled in the agile approach, or more precisely in Scrum. They aim to identify the collaboration related issues that appear in chains of Scrum teams [8]. Three companies, and their corresponding projects, were investigated and are outlined in table 4.4. The interviewees consisted of key personnel as Product Owners, line managers and Scrum Master (Scrum Masters were also considered as developers meaning no other Scrum team members were interviewed).

	Type of Company	**Number of Interviews**	**Number of Chains**
Case study 1	Telecommunications	9 interviews	Two overlapping chains
Case study 2	Insurance	6 interviews	One chain
Case study 3	Insurance	3 interviews	One chain

Table 4.4: Information about case companies.

4.3.1 Issues identified

From the interview rounds six issue areas were identified. These are outlined in table 4.5. The distribution of the different issue areas on the three case studies are further showed in table 4.6. What is interesting is that most of these issues directly or indirectly influences coordination, collaboration and/or communication, and are therefore somewhat coupled (this will be easier to understand when looking at the proposed conceptual model in section 4.3.2). The different issue areas are summarised in their corresponding subsections below with suggested propositions. These propositions are summarised in table 4.7.

Issues description	Issue	Number of quotes
A lack of coordination in the chain	Coordination	124
Mismatches in backlog priority between teams	Prioritization	122
Alignment issues between teams	Alignment	99
A lack of IT chain process automation	Automation	72
Unpredictability of delivery to commitment	Predictability	56
A lack of information visibility in the chain	Visibility	38
	Total	511

Table 4.5: Grounding for each issue.

Issue	Case 1 (%)	Case 2 (%)	Case 3 (%)
Coordination	29	16	18
Prioritization	22	35	17
Alignment	20	22	13
Automation	14	15	15
Predictability	11	5	17
Visibility	4	7	20
Total	100	100	100

Table 4.6: Distribution of issues over the case studies.

Predictability

Predictability has to do with the likelihood that interdependent Scrum team will deliver their feature. If a team does not deliver their feature this will lead to a delay in production. If something is delayed in one sprint it will put more effort on the next sprint introducing major costs from rework, such as retesting and bug fixing, as well as longer time to market.

Coordination

Coordination between Scrum teams was identified as the main issue as can be seen in table 4.5. One of the biggest problems was the focus of the backlog of individual Scrum teams, and not on the end-to-end delivery. This had to do with the perceived lack of influence from management. A problem here is that the Scrum framework supports the coordination theory components on an individual team level, but not for a chain of Scrum teams. Proposition P1 is proposed:

> P1: Embedded coordination practices within and between Scrum teams positively impact delivery predictability.

Prioritization

Priority issues constituted a large portion of the problems encountered in the case studies as well. The main issue in prioritization seems to be mismatching backlog priorities. As each product owner sets his backlog prioritization based on his strategic goals it leads to a mismatch on the front to back chain. This leads to P2:

> P2: Matching priority over the front to back chain positively impacts delivery predictability.

Further prioritization is a way of goal-setting, but this is a component of coordination theory. Therefore, this has to be matched across all Scrum teams in the chain as a single goal leading to P3:

> P3: Matching priority improves front to back coordination practices.

The authors also argue that to improve the priority matching over the front to back chain the priority matching has to be done at the strategic level. This implies implementation of strategic decision. Combining bounded rationality theory and decision making strategies proposition four (P4) is suggested:

> P4: The implementation of decision making strategies improves matched priority setting.

Alignment

Misalignment was also identified in the case studies, such as several definitions of done, difference in sprint cycle lengths and misalignment of test activities and test results between Scrum teams. This misalignment muster unpredictability and delays. P5 is proposed:

> P5: Alignment between Scrum teams positively impacts delivery predictability.

A problem with the Scrum framework is its focus on independent teams. For optimisation, only single teams' processes are looked at. When more teams are introduced a focus has to be shifted towards alignment of work, and how a teams work influences other teams. Using coordination theory again the authors argue that a common shared goal and a coordination mechanism will improve alignment in the chain. This led to P6 and P7:

> P6: Matched priority setting positively impacts the alignment between Scrum teams.

> P7: Coordination practices positively impact the alignment between Scrum teams.

Visibility

The findings also showed that a lack of visibility disabled teams to take appropriate actions, which could easily lead to uncontrollable impediments later in the sprint. If the codependent teams in the chain have visibility over the backlog they will identify the impediments and can therefore take necessary mitigating actions. Control theory suggests that visibility in the chain will have a positive effect on inter-team coordination. This leads to proposition eight:

> P8: Information visibility positively impacts coordination practices.

Automation

Lastly, we have the sixth identified issue area, namely automation. One recognised issue was the lack of backlog status and progress information of codependent teams. This info should be automated to make mitigating activities possible. Through using supply chain management (SCM) literature Vlietland et al. works out the last proposition for the conceptual model:

> P9: Automation of status and progress tracking in the chain positively impacts information visibility.

4.3.2 Conceptual Model

Using all the propositions summarised in table 4.7 and the six identified issue areas a conceptual model is constructed by the authors. They see the conceptual model as a starting point for the development of a governance framework to mitigate the identified issues in chains of Scrum teams. This conceptual model is illustrated in figure 4.2.

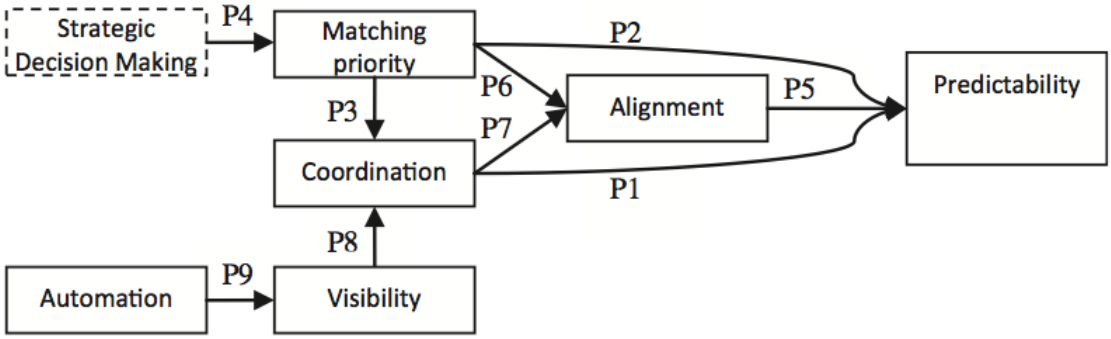

Figure 4.2: Resulting conceptual model.

Proposition ID	Proposition
P1	Embedded coordination practices within and between Scrum teams positively impact delivery predictability.
P2	Matching priority over the front to back chain positively impacts delivery predictability.
P3	Matching priority improves front to back coordination practices.
P4	The implementation of decision making strategies improves matched priority setting.
P5	Alignment between Scrum teams positively impacts delivery predictability.
P6	Matched priority setting positively impacts the alignment between Scrum teams.
P7	Coordination practices positively impact the alignment between Scrum teams.
P8	Information visibility positively impacts coordination practices.
P9	Automation of status and progress tracking in the chain positively impacts information visibility.

Table 4.7: Summary of propositions.

4.3.3 Conclusion

Six issues in chains of codependent Scrum teams were identified and are summarised in table 4.5. From the issue areas and existing theory nine propositions were extracted and are outline in table 4.7. Combining the issues and the propositions the conceptual model in figure 4.2 was created. Coordination is a central part of this puzzle, as can be seen from its direct connection through the propositions to four of the other issue areas (Predictability, Alignment, Visibility and Prioritization/Matching priority), and its indirect connection to the last issue area (Automation) and an added aspect (Strategic Decision Making). It is obvious that the complexity level of coordination is a lot higher in large-scale development, and from Vlietland and Vliet's findings it can be seen that other issue areas need to be addressed together with coordination to achieve a successful and efficient project. It is important to note that this conceptual model needs to be tested in empirical studies to see if it is applicable in practice.

Chapter 5

Discussion

Chapter 6

Conclusion

Chapter 7

Future Work

Bibliography

[1] J. Ågerfalk, B. Fitzgerald, and O. In, "Flexible and distributed software processes: old petunias in new bowls?," *Communications of the ACM*, vol. 49, no. 10, pp. 26–34, 2006.

[2] D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*. Addison-Wesley Professional, 2007.

[3] A. Cockburn, *Agile Software Development*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] D. Batra and W. Xia, "Balancing Agile and Structured Development Approaches to Successfully Manage Large Distributed Software Projects: A Case Study from the Cruise Line," *Communications of the Association for Information Systems*, vol. 27, no. 1, pp. 379–394, 2010.

[5] T. Dingsøyr and N. B. Moe, "Research challenges in large-scale agile software development," *ACM SIGSOFT Software Engineering Notes*, vol. 38, p. 38, Aug. 2013.

[6] M. Paasivaara, "Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?," *ESEM*, pp. 235–238, 2012.

[7] V. O. N. E. Com, "7th Annual State of Agile Development Survey." `http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf`, 2013. [Online; accessed 15-November-2014].

[8] J. Vlietland and H. van Vliet, "Towards a governance framework for chains of Scrum teams," *Information and Software Technology*, vol. 57, pp. 52–65, Jan. 2015.

[9] M. Lindvall, D. Muthig, and A. Dagnino, "Agile software development in large organizations," *Computer*, pp. 26–34, 2004.

[10] E. C. Lee, "Forming to Performing: Transitioning Large-Scale Project Into Agile," *Agile 2008 Conference*, pp. 106–111, 2008.

[11] M. Paasivaara, S. Durasiewicz, and C. Lassenius, "Using Scrum in Distributed Agile Development: A Multiple Case Study," *2009 Fourth IEEE International Conference on Global Software Engineering*, pp. 195–204, July 2009.

[12] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Empirical Software Engineering*, vol. 13, pp. 303–337, May 2008.

[13] S. Freudenberg and H. Sharp, "The top 10 burning research questions from practitioners," *Software, IEEE*, 2010.

[14] K. V. Haaster, "Agile in-the-large: Getting from Paradox to Paradigm." 2014.

[15] T. Dingsøyr, T. E. Fægri, and J. Itkonen, "What is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development." 2013.

[16] D. Reifer, F. Maurer, and H. Erdogmus, "Scaling Agile Methods," *IEEE Software*, vol. 20, pp. 12–14, July 2003.

[17] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[18] A. Borjesson and L. Mathiassen, "Successful Process Implementation," *Software, IEEE*, 2004.

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

[20] Dr. Royce, Winston W., "Managing the Development of Large Software Systems," 1970.

[21] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods - Review and analysis," Tech. Rep. 478, VTT PUBLICATIONS, 2002.

[22] Takeuchi, Hirotaka and Nonaka, Ikujiro, "New New Product Development Game," 1986.

[23] J. Sutherland, "Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies," vol. Vol. 14, No. 12, Dec. 2001.

[24] M. Cohn, "Advice on Conducting the Scrum of Scrums Meeting." `https://www.scrumalliance.org/community/articles/2007/may/advice-on-conducting-the-scrum-of-scrums-meeting`, 2007. [Online; accessed 11-December-2014].

[25] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Comput. Surv.*, vol. 26, pp. 87–119, Mar. 1994.

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

[27] J. Schnitter and O. Mackert, "Large-scale agile software development at sap ag," in *Evaluation of Novel Approaches to Software Engineering* (L. Maciaszek and P. Loucopoulos, eds.), vol. 230 of *Communications in Computer and Information Science*, pp. 209–220, Springer Berlin Heidelberg, 2011.

[28] I. O. Robert L. Nord and P. Kruchten, "Agile in distress: Architecture to the rescue." 2014.

[29] D. E. Strode, B. G. Hope, S. L. Huff, and S. Link, "Coordination effectiveness in an agile software development context.," in *PACIS* (P. B. Seddon and S. Gregor, eds.), p. 183, Queensland University of Technology, 2011.

[30] C. De O. Melo, D. S. Cruzes, F. Kon, and R. Conradi, "Interpretative case studies on agile team productivity and management," *Inf. Softw. Technol.*, vol. 55, pp. 412–427, Feb. 2013.

[31] N.-D. Anh, D. S. Cruzes, and R. Conradi, "Dispersion, coordination and performance in global software teams: A systematic review," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, (New York, NY, USA), pp. 129–138, ACM, 2012.

[32] T. Dingsøyr and Y. Lindsjørn, "Team performance in agile development teams: Findings from 18 focus groups," in *Agile Processes in Software Engineering and Extreme Programming* (H. Baumeister and B. Weber, eds.), vol. 149 of *Lecture Notes in Business Information Processing*, pp. 46–60, Springer Berlin Heidelberg, 2013.

[33] L. A. Goodman, "Snowball sampling," *Ann. Math. Statist.*, vol. 32, pp. 148–170, 03 1961.

[34] K. S. Khan, R. Kunz, J. Kleijnen, and G. Antes, "Five steps to conducting a systematic review.," *Journal of the Royal Society of Medicine*, vol. 96, pp. 118–121, Mar. 2003.

[35] M. Paasivaara and C. Lassenius, "Communities of practice in a large distributed agile software development organization – case ericsson," *Information and Software Technology*, vol. 56, no. 12, pp. 1556 – 1577, 2014. Special issue: Human Factors in Software Development.