Research Project (TEST)

TEST

December 4, 2014

Author:
Espen Andreassen

Advisor:
Torgeir Dingsøyr

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. Theories, literature and frameworks 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 efficiency achieved in large-scale agile 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 subsection 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 research project only aims to cover the described research question.

Further, the research project does not aspire to introduce a brand new theory regarding the combination of large-scale, agile software development, coordination and efficiency. 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 search 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.

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,

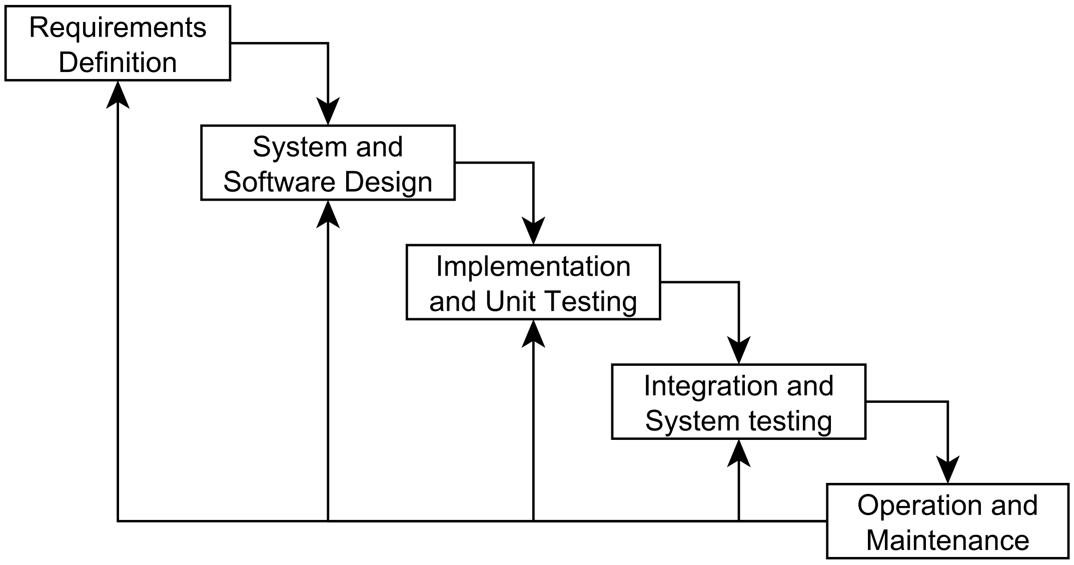
Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System testing
Operation and Maintenance

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.

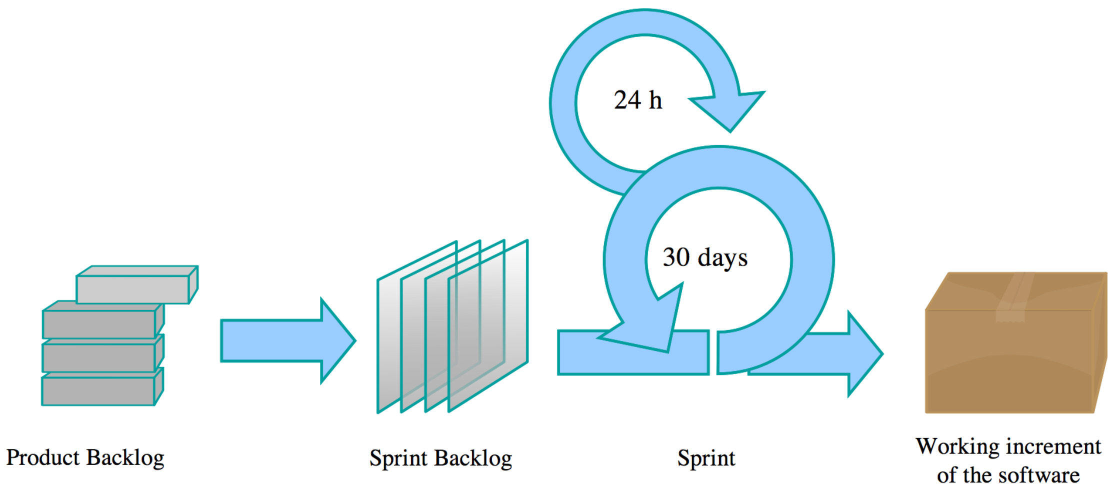

Figure 2.2: The Scrum cycle.

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.

2.2 Coordination

2.3 Large-scale

What is actually "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)) [23].

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 [24].

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 Productivity and Efficiency

Chapter 3

Method

Chapter 4

Results

Chapter 5

Discussion

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. 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.

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