NTNU

Norwegian University of
Science and Technology

Project Management in Agile Software Development

An empirical investigation of the use of Scrum in mature teams

Joachim Hjelmås Andersen

Master of Science in Computer Science
Submission date: June 2009
Supervisor: Torgeir Dingsøyr, IDI

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

Problem Description

Agile software development

A range of new methods for software development has been called "agile", such as eXtreme Programming (XP), Scrum, Rational Unified Process (RUP), lean development, crystal clear and crystal orange. These methods have been proposed by practitioners, based on what has worked well for them. One of the most popular agile development methods today is Scrum, which has a strong emphasis on project management.

There are, however, few studies that show how Scrum works in different development projects, while the existing studies tend to focus on the introduction of Scrum. The subject for this thesis is to investigate what characterizes project management in mature Scrum teams in the software development industry through an empirically supported analysis.

Assignment given: 19. January 2009
Supervisor: Torgeir Dingsøyr, IDI

Abstract

Project management is generally a challenge in software engineering and during recent years many are of the opinion that agile development may be a solution. As a result, agile development, which focuses on individuals and interactions, working software, customer collaboration, and the ability to respond to change, has gained much popularity and is widely adopted in the industry. However, few studies that examine project management in agile development currently exist. As a consequence, this report investigates the existing practices employed in projects running Scrum through a literature survey of empirical studies and compares the findings with an empirical multi case study, based on semi-structured interviews, in order to determine what characterizes project management in mature Scrum teams. This examination was carried out in relation to a framework for teamwork, as agile development is a team effort and thus needs to be managed accordingly. It was found that Scrum is to a large degree modified in agile environments in the industry, as several practices identified are not a part of Scrum. These practices are employed in order to complement Scrum so that it fits the particular organization, project or both. Examples of such practices include governance meeting, reference group, pre-planning initiatives, sprint celebration, daily Scrum of Scrums and Meta Scrum.

Implications for research and practice are presented.

Keywords: Agile, Scrum, mature teams, project management, team leadership, case study, software engineering.

Preface

Ever since I first started programming, I have been intrigued by iterative development. The reason being, that I have always worked in an iterative manner, starting with the very basics and slowly improving my work step-by-step. In regards to software development, I find it essential to be able to improve a product by adapting to the customer's needs through continous feedback. Consequently, when Scrum came to my attention I was thrilled and after learning more about Scrum through my summer internship last year, I was determined to find a way to further explore this topic. The process started by contacting Torgeir Dingsøyr in January and together we determined a suitable approach.

This in-depth study report was written by Joachim Hjelmås Andersen in January through June 2009. The project was proposed by Torgeir Dingsøyr, and has been developed in the section of Design and Use of Information Systems in the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Acknowledgements:

I would like to thank my advisor Torgeir Dingsøyr for excellent guidance, valuable discussions, insights and continuous feedback throughout the whole project. In addition, I would like to thank the representatives, from the four different companies involved in this report, for taking the time to let me interview them. The feedback from these individuals has been very informative and of great value to me and my project. Thank you.

Trondheim, 15.06.2009

_____

Joachim H. Andersen

Table of Contents

List of Figures

List of Tables

1 Introduction

This report materializes a semester long project which delves into a literature study and an empirical investigation considering project management practices in software development projects involving mature Scrum teams in Norwegian companies. This chapter serves as an introduction to the project and starts off by providing the motivation behind researching this topic. Then the problem definition is stated and the scope of the report is defined. Finally, an outline of the entire report is presented.

1.1 Motivation

Earlier most companies followed a plan-driven software development methodology with a low degree of customer involvement and strong emphasis on documentation (Maurer & Melnik, 2006). Generally, software development projects tend to be problematic, and it is reported from both Gartner[1] and Standish[2] periodic surveys that about 75% of all projects are challenged or fail to deliver according to agreed commitments. On the other hand, these surveys remain controversial as skeptics claim that they are based on a biased selection process from an unknown population. Nevertheless, an important challenge in projects is to identify and deal with shifting customer needs. Consequently, to cope with the ever changing business environments and customer requirements agile development has emerged. According to Digital Focus Agile 2006 Market Survey[3], 46% of mid-sized companies are adopting agile practices company-wide, while only 12% of large companies are doing the same, however 44% of these organizations are using agile practices on a project. Furthermore, in a survey performed by (Parsons, Ryu, & Lal, 2007), they found that nearly 16% stated that they were using multiple agile methods. Moreover, they found that Scrum, which is an agile project management approach, is one of the most used agile methods and is now widely adopted in the industry.

[1] Gartner Research Notes #TU-11-0029 (A Project Checklist) and #SPA-13-5755 (IT Portfolio Management and Survey Results). Similar survey results in: Vanderwicken Financial Digest, Standish Group, http://www.iqpc.com

[2] Standish Group 1994, The CHAOS Report, 1-2, : http://www.standishgroup.com

[3] http://www.infoq.com/news/Digital-Focus-Unveil-Survey-2006

However, there is a lack of information regarding how Scrum is practiced in companies. As seen in (Dybå & Dingsøyr, 2008), only one empirical case study of Scrum existed up until 2005, thus additional such studies are needed, in order to understand how Scrum is adopted by the industry.

Project management is a very important issue as projects will often have changing requirements, which in turn leads to modifications to scope and budget and no or lack of control dramatically increases the risk of failure. Consequently, stories of projects that have run out of time, over budget or both have often been seen in the press (Maylor, 2001). This can be seen from Dag-Brücken (Jewels, 2003), where lack of project management and managers without the necessary skills were appointed, which lead to the downfall of a major project in Taiwan. A wide range of such examples is available in the literature. Additionally, it might prove difficult to attain 100% dedication to a project from all the available staff, since people are often working on different projects at the same time (Elwer, 2008). Furthermore, it can be complicated to monitor how the project is progressing and what the team members are currently working on. It is also reported in (Dingsøyr, Dybå, & Abrahamsson, 2008), that focus should be placed on management-oriented approaches, such as Scrum, as this has both impact and potential for the industry, but has received very little attention form the research community. Additionally, the need for a comparison with traditional plan-driven project approaches through a qualitative analysis to further consider the implications of agile practices is also argued for by (Pikka-rainen, Haikara, Salo, Abrahamsson, & Still, 2008).

Due to project management being a very hard discipline to master and the lack of literature addressing this knowledge area, there currently exists a lack of agreement regarding which practices are appropriate or not. This has left researchers and practitioners alike asking: “What characterizes project management in mature Scrum teams?” This report is motivated to answer this question through an empirically supported analysis of mature Scrum teams in today’s software development industry. Currently existing practices are examined and by understanding how the companies investigated have adapted the project management practices in Scrum as the organization and the teams matured, important instruments, techniques and possible trends can be documented and caveats identified. This will hopefully provide scientists with specific areas for further research as

well as assist and inspire companies considering introducing Scrum, and also companies in need of guidance after having initiated Scrum.

The research methods employed in this project has been to investigate empirical studies in the literature and also Norwegian software development companies performing Scrum through semi-structured interviews.

1.2 Problem Definition

This work seeks to add to the pool of existing Scrum studies. Since, practice is often ahead of research, in the software field, much can be learned from examining good practice (Fitzgerald, Hartnett, & Conboy, 2006). Thus, the approach employed was to study mature Scrum development teams in Norwegian software companies to determine if problems related to leadership, as pointed out in (Moe, Dingsøyr, & Kvangardsnes, 2009), in agile development has been overcome as the methods became more established within the company. A literature study and an empirical multi case study has been carried out in order to compare how practice separates itself from theory and to identify differences in Scrum practices as the teams become more mature.

Based on the above, the objective of this research was to investigate:

- What characterizes project management in mature Scrum teams?

Mature in this context, is defined as teams consisting mainly of individuals who has been using Scrum for 1.5 years or more. Moreover, it is not required that the whole team is mature, as team members more experienced in Scrum, and thus more mature, will most likely take control and guide the other team members through the process and introduce the practices found needed. Neither, is it required that the project the team is working on has been running for 1.5 years or more, since the individuals in the team may have valuable Scrum experience from previous projects. It should be noted, however, that immature teams (teams who have recently started working on their first Scrum project) will not be excluded from this report as a basis of comparison is needed. In addition, interesting practices might also be discovered from such teams.

In regards to the research question, it is relevant to investigate the practices employed in various projects; how problems are solved, how teams cooperate, what motivates them and

the meeting practices used. Additionally, it is of interest to examine how planning is carried out, how progress is measured and what information that affects the teams. It may also be beneficial to look into how the employed software methodology has been adapted to fit the project or company's needs as time progressed and what practices have been added to complement Scrum. Studies investigating cherry-picking of key practices and combinations of software development methods i.e. Scrum and XP, are also argued for by (Parsons et al., 2007)).

1.3 Scope of Report

In order to determine what characterizes project management in mature Scrum teams, a framework to employ as a lens when extracting important practices was needed. The framework chosen was based a selection of five of the behavioral makers for team leadership, which is defined as one of the "Big Five" of teamwork in (Salas, Sims, & Burke, 2005). Further on, I chose to examine the available literature in order to identify essential practices related to project management when running Scrum. These findings were then contrasted to the practices discovered in the industry through an empirical multi case study, based on semi-structured interviews. As the literature is scarce in regards to mature Scrum team practices, the goal was to determine how Scrum is adapted over time, which practices that are needed to complement Scrum after the initial phases and what the trends seem to be for the future by comparing my findings. I also aimed towards identifying sparsely documented areas of Scrum, in order to signal where additional research should be directed. Further on, it should be noted that I will not treat project management activities such as contract administration or budgeting which are not covered by team leadership, as it is defined in (Salas et al., 2005), in this report.

1.4 Report Outline

The remaining parts of the report are structured in the following way:

2: Theory

This chapter starts out by presenting plan-driven software development in section 2.1, while section 2.2 describes agile software development and the subsection 2.2.1 delves into Scrum in more detail. Further on, section 2.3 provides a look at project management in

Scrum and section 2.4 offers a comparison between project management in Scrum and plan-driven software development. Finally, section 2.5 presents the framework that will be used to analyze project management in studies of Scrum.

3: Research Methods and Design

The research methods used were a literature survey of empirical studies and an empirical multi case study. In this chapter a description of the data collection is given in section 3.1 whereas subsection 3.1.1 gives an in-depth explanation of how the literature study was performed, while subsection 3.1.2 details the empirical multi case study. Further on, section 3.2 provides a walkthrough of how the data analysis was carried out.

4: Literature Study Results

Throughout the opening section 4.1, in this chapter, the studies selected during the literature study are categorized. Then the results are analyzed through the lens of the framework, presented in section 2.5, throughout section 4.2 – 4.6.

5: Empirical Multi Case Study Results

This chapter starts out by presenting the projects and the companies, which were in charge of the projects, in section 5.1. The rest of the chapter, that is section 5.2 – 5.6, is devoted to evaluating the findings from the empirical case studies according to the framework, given in section 2.5.

6: Discussion

This chapter asses the findings and results from chapter 4 and 5 based on the framework, established in section 2.5. In the end, threats to validity of the report are treated.

7: Conclusion

This chapter concludes the report. Section 7.1 summarizes the findings and contributions in regards with the framework. Finally, implications for practice and indications of future trends for Scrum in the industry as well as directions for further work are presented in section 7.2.

2 Theory

This chapter presents traditional plan-driven software development, agile development and Scrum in general, but will keep the main focus directed towards project management aspects. Scrum will be devoted the most attention as it emphasis project management in agile development. Then project management for Scrum in particular will be addressed in more depth and what Scrum, as a software development method, does not offer will be high-lighted. Further on, a comparison between project management in Scrum and plan-driven software development is given and then I will present the framework that I will use when analyzing my findings.

2.1 Plan-Driven Software Development - the Waterfall Model

In the traditional waterfall methodology (Royce, 1987) a rigorous plan for the given project is developed before embarking on the actual project, and tightly followed throughout the development life-cycle. The requirements collection and the analysis are performed at the start of the project. The design is also fully developed before starting to code and is expected, along with the requirements, to remain unchanged during the rest of the project. This illustrates the core of the methodology, being that each of the seven phases (see figure 2.1) of the project life-cycle is completed before the next commence and the products from the earlier phases, i.e. documents produced, serve as input for the next ones. This cascading effect of previous results is analogous to a waterfall, hence the name.

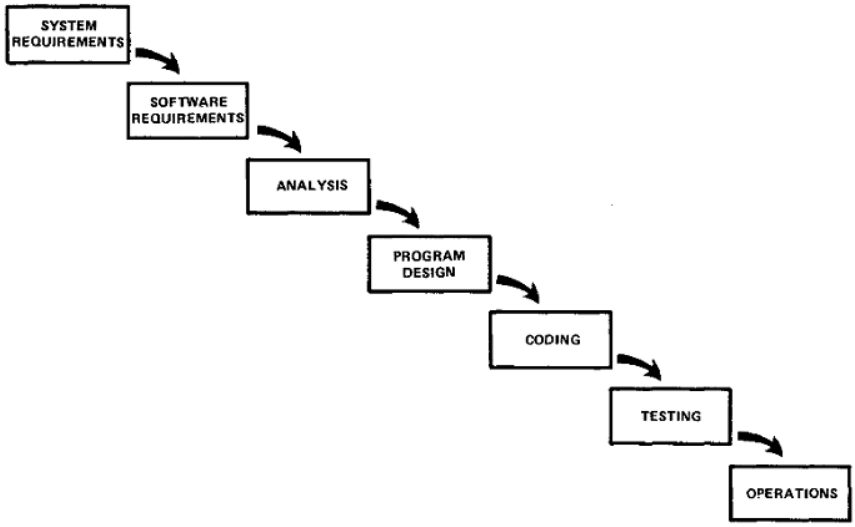

Figure 2.1: The seven phases of the project life-cycle as it is defined in the Waterfall Model

Source: (Royce, 1987)

Furthermore, teams will typically consist of people with skills from the same domain and team members will be assigned to specific phases of the project (Hawryszkiewycz, 2000). Furthermore, throughout the whole project, documentation plays a significant role in order to ensure a high degree of repeatability (Maurer & Melnik, 2006).

2.2 Agile Software Development

Agile software development (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003; Dybå & Dingsøyr, 2008) is a common term for a range of development methodologies. Furthermore, agile development focus on providing high customer satisfaction through quick delivery of working software, active participation of concerned stakeholders, and by creating and leveraging change (Highsmith, 2002). By actively involving stakeholders, communication and cooperation within projects are improved, which increases the likelihood of successfully fulfilling customer needs. Continuous tight communication with the stakeholders and especially within the project team, also contributes to limiting the formal documentation to the essentials, thus allowing the developers to focus on producing working software instead of documentation. The agile development methods separate themselves from the traditional plan-driven and document-driven methodologies (Sørensen, Jense, & Holm, 2008) by, among other things, being people-centric, exercising facilitation and not controlling management as well as focusing on a more hands-on project

management style (for a more comprehensive comparison see (Nerur & Balijepally, 2007)).

Further on, agile development involves frequent inspection through short development iterations and a high degree of testing as well as leaning towards a flat organizational structure by forming self-organizing cross-functional teams and high process adaptability throughout the whole project. When summed up, the core of the typical practices promoted by the agile paradigm are:

> "**individuals and interactions** over processes and tools,
> **working software** over comprehensive documentation,
> **customer collaboration** over contract negotiation, and
> **responding to change** over following a plan." [4]

Moreover, as given by the agile manifesto; "while there is value in the items on the right, the items on the left are valued more." In short, the core of the agile methods is to support continuous adaptation of the development with the needs and expectations of the customer. When considering what the available literature reports, regarding meeting customers' needs and expectations, (Parsons et al., 2007) indicates, through a survey, that agile methods enhance software development projects in terms of stakeholder satisfaction, quality and productivity without a significant increase in cost.

The agile element of having self-organizing teams will according to Tata & Prasad (Tata & Prasad, 2004), alleviate the project manager from some of his duties by giving the team more authority and power to make decisions. As a consequence, problem solving is performed faster and more accurately while the team takes on more ownership of the process. Moreover, the principle of self-organizing teams is largely congruent with leadership theory Y (McGregor, 1960). This is manifested by the fact that according to theory Y; managers assume that the employees are ambitious, self-motivated and anxious to receive greater responsibility as well as exercise self-control. Furthermore, higher productivity can be achieved by offering the employees freedom and trust them to do their very best. Thus the agile principles allows freedom within limits, conditions under which

[4] Beck, K et al. Manifesto for Agile Software Development, 2001. Online: http://www.agilemanifesto.org

the urge most people have to do their best will flourish given the assumption that the satisfaction of doing a good job is a motivation in itself.

Traditional methodologies aim to make software development into a repeatable, predefined and predictable process[5]. However, agile methods do not have the same focus on repeatability, or the other two aspects mentioned for that matter, as the plan-driven development paradigm is known for. It is considered much more important that e.g. a process, system or project is agile and fits its purpose, than being built in order to be recycled at a later stage. Moreover, it is a common perception that by following this principle, both time and resources are saved by not having to make every component as general and reusable as possible. Software development implies so many changes both within a project and across projects that there is only a minimal gain in reusing existing software solutions. It is better to reuse the knowledge gained from previous projects, learn from past mistakes and develop systems from scratch, than to spend time and resources on modifying an old system to a new project. That is why development strategies and practices need to reflect the uniqueness of projects, thus agile methods focus on translating principles rather than utilizing best practices.

Agile software development methodologies, methods and techniques are gaining popularity and an analysis performed by (Parsons et al., 2007) shows that Scrum (Schwaber & Beedle, 2001) and eXtreme Programming (XP) (Beck & Andres, 2004) are the most commonly used pairs of agile methods. Scrum is an agile framework for project management designed to make system development more effective (Moe, Dingsøyr, & Dybå, 2008; Schwaber & Beedle, 2001), while XP is more concentrated on implementation and offers concrete guidelines to agile development practice, as seen in figure 2.2. XP is actually a compilation of well-known software engineering practices and among the twelve XP practices mentioned in (Abrahamsson et al., 2003), pair programming, continuous integration and test-driven development are very much referred to techniques as seen in (Mann & Maurer, 2005) where Chris Mann chose to introduce these particular techniques in the project. This may indicate the importance of employing these practices in software development projects. According to (Dybå & Dingsøyr, 2008), XP is also the

[5] Cohen, D., Lindvall, M., and Costa, P. (2004). An introduction to agile methods. Advances in Computers, 62, p.1-66.

most widely researched agile method. Further on, when considering figure 2.2, six agile methods support project management, among these, Scrum is the only method that does not exclusively rely on abstract principles, but also offer some concrete guidance on project management. After examining Scrum and XP's characteristics it is evident that by combining them they can complement each other as they have their strengths on different levels. As a consequence, several agile software companies as Intel Shannon (Fitzgerald et al., 2006) use Scrum and XP in concert, by employing practices such as pair programming (a practice known from XP) when implementing, in order to take advantage of their synergies and thereby achieve the best results.

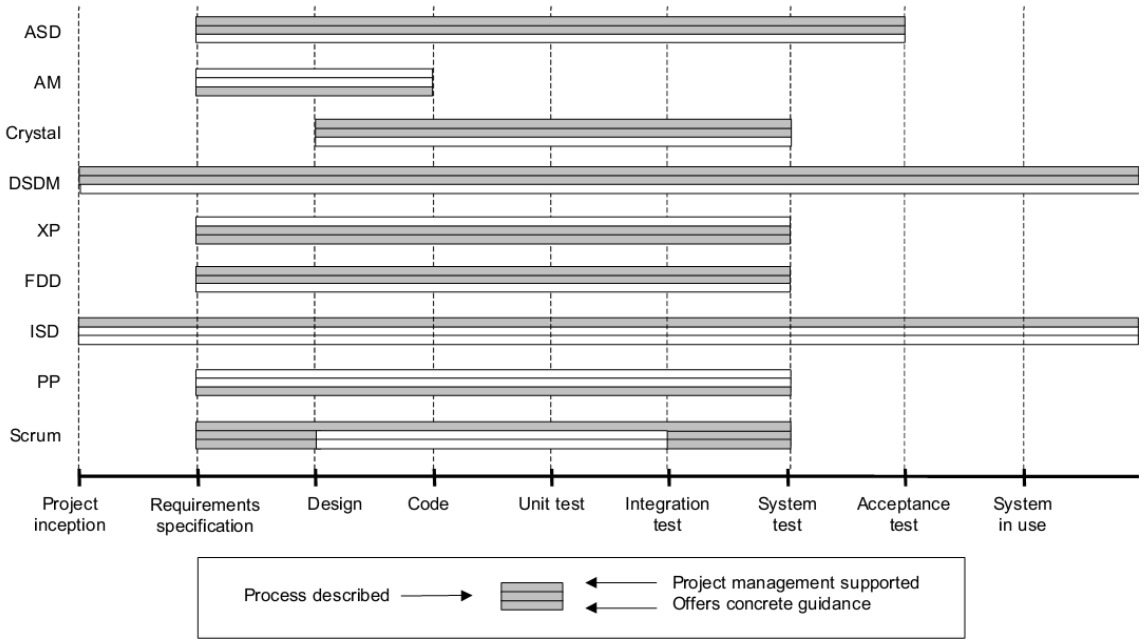

Figure 2.2: Comparison of project management support in different methods and methodologies

Source:(Abrahamsson et al., 2003)

Although the use of agile development has gained increasing popularity in the recent years, its practices and procedures often lack empirical backing, as most existing publications are written by practitioners or consultants (Abrahamsson et al., 2003; Dybå & Dingsøyr, 2008). Furthermore, as Scrum places its emphasis on project management, is currently much used and few research studies exists, as pointed out in (Dybå & Dingsøyr, 2008), Scrum and other relevant practices used in the industry will be the focus of my attention through the rest of this report.

2.2.1 Scrum

The Scrum methodology was strongly influenced by a Harvard Business Review article[6] from 1986 on practices associated with successful new product development groups, in which the term "Scrum" was introduced. The term "Scrum" was actually borrowed from rugby, where: "[A] Scrum occurs when players from each team huddle closely together...in an attempt to advance down the playing field" (Highsmith, 2002). This closely resembles the effort of the self-organizing development teams when working closely together attempting to reach a common objective, while the project manager has a coaching role, which explains why the Scrum analogy is used.

My discussion will mainly revolve around Scrum and its practices and how they relate to project management. As seen from (Schwaber & Beedle, 2001), Scrum is often adopted in project management situations with rapidly changing environments where it might prove difficult to plan ahead. To meet such conditions, Scrum utilizes short iterations (sprints), continuous feedback loops fostered through daily stand-up meetings and sprint retrospectives. Due to this, Scrum is often introduced in order to improve a company's ability to deliver iteratively with high quality in a timely manner and as a secondary effect, team communication and fellow feeling is greatly improved. In a way, Scrum moves focus away from the old project-centric model towards a continuous product development model. Furthermore, as there are no project with a beginning, middle and end, there is no traditional project manager role. Instead a *product owner* and a stable self-managing team collaborate across sprints, while typical "project management work" is handled by the team and the business owner, who is an internal business customer or from the product management. The team itself is lead by the *scrum master*.

Scrum master: The *scrum master* encourages communication, provides creative input, guides the skills development of team members, facilitates meetings and takes on a coaching role, while supporting collaboration between all stakeholders involved. The *scrum master* may use Socratic questioning in order to initiate thinking processes within the team and thereby facilitating the discovery of the solution to a problem. It is the *scrum*

[6] Takeuchi, Hirotaka and Nonaka, Ikujiro. (January-February, 1986). "The New New Product Development Game." Harvard Business Review.

master's role to assist and remove impediments for the team in order to help them work more effectively. This is also congruent with the manager role known from theory Y (McGregor, 1960), whereas a manager should try to remove the barriers that prevent workers from fully actualizing their potential. Furthermore, the *scrum master* will not decide how the team will work or who's doing what, but instead act as a facilitator or mentor for the team. However, he/she will work closely with the *product owner* in order to prioritize the work to be done by the *scrum team* in the next iteration (sprint).

Sprint: A sprint is a short development iteration that typically lasts for 2-4 weeks. Each sprint starts by setting goals and determining the functionality to be included in the upcoming sprint-release and a hard deadline for when the sprint ends. Teams are encouraged to choose one sprint-duration and not change it (too often). By having a consistent sprint-duration, it allows to the team to learn how much it can accomplish more quickly, which is helpful both when estimating and for longer-term release planning. Additionally, it gives the team a work rhythm, which is commonly referred to as the "heartbeat" of the team.

Scrum team: The teams are typically small, consisting of 5-9 developers with cross-functional skills. Each team is self-organizing (self-managing) with a high degree of autonomy and accountability, and works closely together to develop potentially shippable software for each sprint. The team is in charge of delegating the tasks to be done within each sprint and how to organize themselves in relation to the work at hand. The nature of self-organizing teams encourages involvement and participation. A consequence of the team members' deep involvement in the distribution of tasks, decision-making and team problem solving, may be that they become more committed to their work and develop a greater sense of ownership to the product. Furthermore, this may foster increased creativity and lead to more effective development, teamwork and a higher quality product.

The principles that are laid down for a Scrum team closely resemble how high-performance teams are defined by Katzenbach & Smith[7]; "a high-performance team consists of a small number of people with complementary skills equally committed to a common purpose, goal and working approach for which they hold themselves mutually

[7] Katzenbach, J., & Smith, D. (1993). The Discipline of Teams. *Harvard Business Review, 71* (Mar-Apr), p.111-120.

accountable. The team members are deeply committed to one another's personal growth and success. That commitment usually transcends the team which significantly outperforms all other like teams and reasonable expectations given its membership." Furthermore, according to Katzenbach & Smith, high-performance teams are the best and most mature of the five types of teams that they define, which are; working group, pseudo team, potential team, real team and high-performance team. Based on Katzenbach & Smith's description of high-performance teams, it is evident that it is not trivial to develop such a team, thus illustrating why it can be challenging to execute Scrum properly and makes it understandable that many aspects of Scrum is not possible to attain until the team has matured in regards with the process.

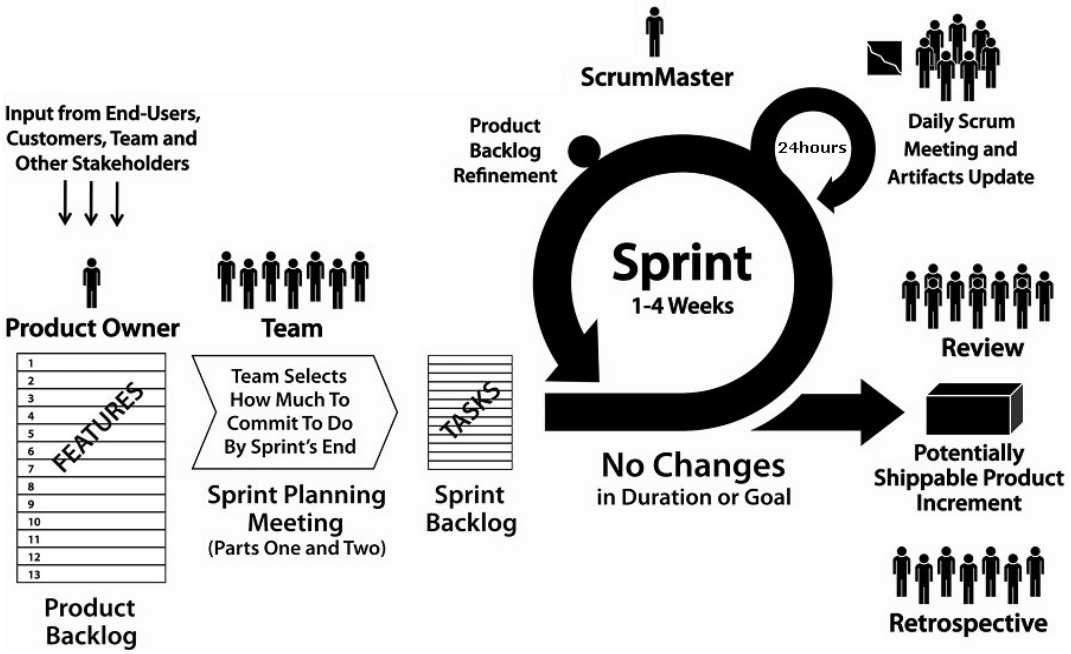

Figure 2.3: The Scrum process.

Source: (Deemer, Benefield, Larman, & Vodde, 2009)

Sprint planning: Each sprint starts with a short planning meeting, as seen in figure 2.3, which is normally considered to have two parts. In the first part of the planning meeting, the *product owner* and the team discuss the top priority items in the product backlog and their context in order to achieve a common understanding of what to develop. Then the definition of done for the items to be developed are reviewed. The question that needs to be answered is; "What is done?" or "When do we know that we are done?" In order to answer this question, acceptance criteria for the work to be completed needs to be

established. For instance, done can mean that the work is developed according to code-standards, reviewed, tested with 100% correctness and has more than e.g. 80% test coverage for the code, integrated and documented.

After this, the *product owner* is free to leave, however, he/she must be available by e.g. phone if needed during the next part of the planning meeting. In the second part of the sprint planning, the team performs detailed planning for how to implement the backlog items, which were chosen by the team. During this phase, the items to be developed are fleshed out on story cards and placed on the wall/task board (or flip-over, whiteboard etc.). The story cards and the wall (task board) as well as the importance of these artifacts are explained in great detail in (Sharp, Robinson, & Petre, 2008). The work required to finish each story card is often estimated by the team through an exercise called the planning game or planning poker. See (Kniberg, 2007) for an in-depth discussion and figure 2.4 for an illustration.

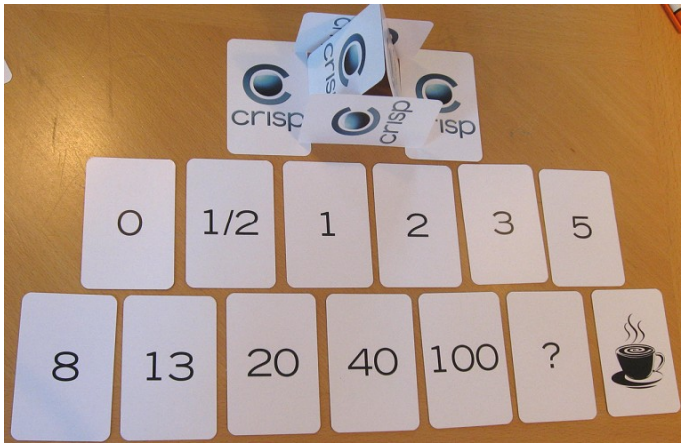

Figure 2.4: Planning poker

Source: (Kniberg, 2007)

Once all the stories are estimated and placed on the wall, the team will distribute the tasks at hand. While working on a story its appurtenant story card will normally be removed from the wall and a placeholder may be kept on the wall instead.

Daily stand-up meeting: Every day during the sprint, typically during the first few hours, a brief stand-up meeting is held (in front of the wall) (Rising & Janoff, 2000). This meeting should, according to (Schwaber & Beedle, 2001), last for maximum 15 minutes and

everyone attending should stand during the meeting in order to ensure that it remains short. The agenda for this meeting is for every team member to answer the following questions:

- What have you done since the last meeting?
- What are you planning to do towards the next meeting?
- What impediments are threatening your work, if any?

The main purpose of this meeting is, as indicated by (Schwaber & Beedle, 2001), to quickly get an overview of the project status. In order to keep the meeting short, any problems identified will not be solved during the meeting. However, the *scrum master* will take note of the problems and try to solve these or facilitate any changes required to help the team, after the meeting. Once the team is aware of existing problems the necessary resources may be allocated to fix the problem i.e. two people teaming up to solve a difficult task.

Product owner: The *product owner* is responsible for maximizing return on investment (ROI – highest-business-value & lowest-cost) by identifying product features and translating these into a prioritized list where the top most are chosen to be implemented for each sprint and put in the sprint backlog. The *product owner* will continually refine and re-prioritize this list. The *product owner* is selected collectively by the *scrum master*, the customer and the management (Moe et al., 2009). He/she represents the customer in the project and will either be an envoy from the customer or someone working within the company in close dialog with the customer. The *product owner* may also participate when estimating the effort required for the particular backlog items as they are fleshed out into tasks to be developed during the sprint planning.

Product backlog: The product backlog is a prioritized list of all high-level product and business requirements, technical features, bug fixes, procedures, updates/upgrades, major patches or other enhancements for the final product which are often written in the form of user stories (more on this further down). The product backlog is constantly being prioritized by the *product owner* according to the highest value-to-effort, in other words quick wins is prioritized, as seen in figure 2.5.

User Story	Value	Effort	Value / Effort	Priority
User Story 1	8	13	0,615	4
User Story 2	8	5	1,6	3
User Story 3	13	3	4,33	1
User Story 4	20	8	2,5	2
User Story 5	20	?	?	N/A

Figure 2.5: Example of a product backlog.

The product backlog is utilized in order to facilitate communication and management of requirements, features and project task dependencies in the software development teams and is regularly updated to reflect the changes necessary to satisfy the customer. The product backlog is normally maintained and prioritized by the *product owner*, although several stakeholders such as the customer, the team, maintenance personnel, management and the *scrum master* may request the making of product backlog items as well. Moreover, since the significance of the different tasks to be done in the project varies, the product backlog helps the team and the *scrum master* to make the *product owner* constantly prioritize what is most important in order for a clear focus to be sustained.

Figure 2.6, on the next page, illustrates a product backlog and as seen from the figure a product backlog may (this is normally the case) consist of several sprint backlogs. In the figure, this is illustrated by Iteration 4.9.2003 and Iteration 1.11.2003. The first-mentioned sprint backlog is expanded, thus showing all the items that are to be completed during this particular sprint.

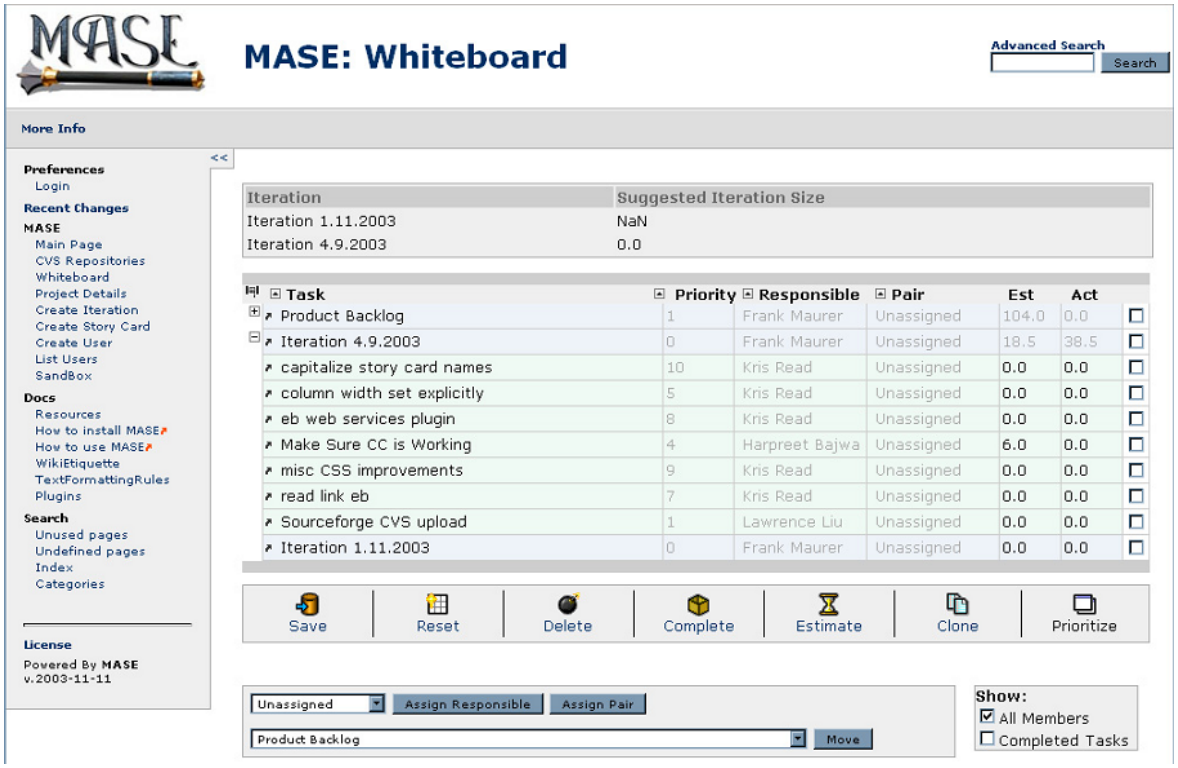

Figure 2.6: MASE project planning whiteboard - example tool for maintaining the product & sprint backlog.

Source:(Chau & Maurer, 2004)

Sprint backlog: The sprint backlog is a detailed plan describing what should be done by each team member and consists of a subsection of the product backlog involving the requirements that are selected for implementation in the current sprint. The sprint backlog should not be modified during the sprint (Moe & Aurum, 2008), unless under extreme circumstances when some kind of sudden critical task emerges or a fatal error occurs in i.e. a previous release and debugging is required or if the team does not manage to carry out the work as planned.

Burndown chart: A useful tool when managing projects in Scrum is the use of a burndown chart. The purpose of this chart is to show progress and highlight discrepancies in regards to the initial time estimates that were produced during the sprint planning. The burndown chart is typically updated according to the tasks completed since the last meeting during the daily stand-up meeting. By closely monitoring the burndown chart, it is possible for the *scrum master* and the team to identify at an early stage when they are off track or when additional resources are required in order to keep the deadline. In many cases the chart may

actually serve as a motivating factor as people tend to be rather pessimistic when making the initial estimates, thus when the team discover that they are ahead of schedule this might have a positive effect on the work environment within the team. The actual chart is a graph with the total remaining amount of story points on the y-axis and the time used so far on the x-axis. An illustration of a burndown chart can be seen in figure 2.7 below.

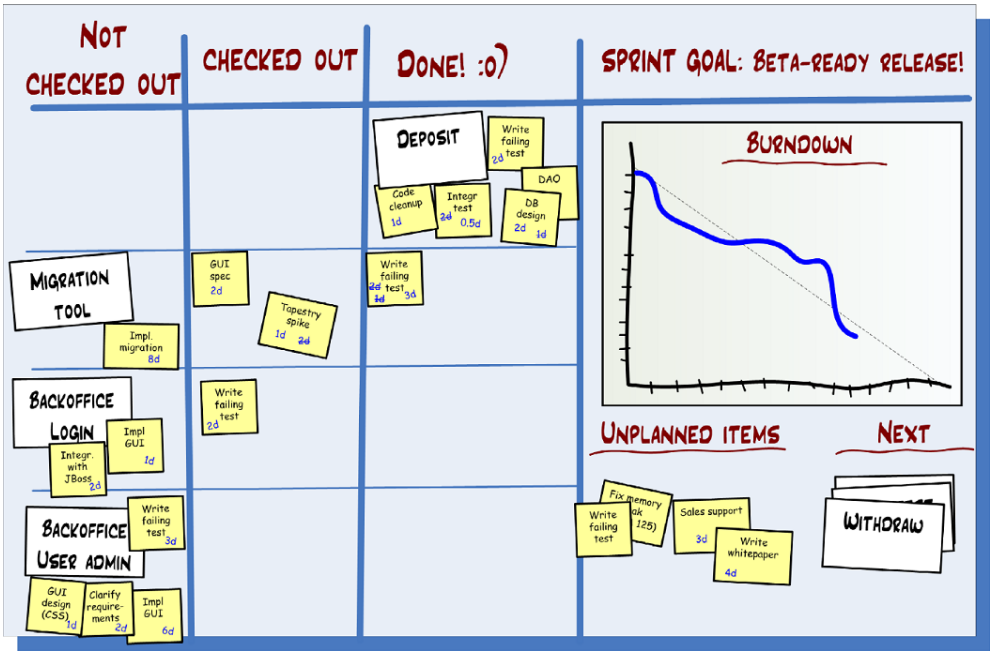

Figure 2.7: A typical wall in a team room, with the burndown chart to the far right.

Source: (Kniberg, 2007)

Sprint Review: After each sprint a sprint review is held where the team demonstrates the results of the sprint to the *product owner* in order to attain the final approval. This is an opportunity for the team members to show what they have been working on, which may boost self-esteem and may serve as a reward in itself. It is customary to finish each demo session with applause. In addition, the demonstration provides the other participants, in the meeting, with an overview of the progress and the work done. An important part of the review is also the conversation between the team and *product owner* in regards to the results presented. The people attending this meeting are normally the team, the *scrum master*, the *product owner* and any others that take interest in the sprint output.

Sprint retrospective: Each sprint ends with a review and then a retrospective meeting where the team reflects on the finished sprint in order to be more efficient and effective in the next. The sprint review involves inspecting and adapting in regards with the product,

while the sprint retrospective is about improving the process through inspection and adaptation. Consequently, important questions to be answered during the retrospective meeting are:

- What went well?
- What went wrong?
- What can we improve for the forthcoming sprint?

The people attending the sprint retrospective are typically the team and the *scrum master,* while the *product owner,* customer representatives or other teams may attend if they wish. As with the sprint review, it may be beneficial for other teams to attend in order to get an overview of what this particular team is doing and learn from their experiences.

Scrum of Scrums: A weekly or biweekly meeting that usually involving one representative from each team (chosen by the team) and the all the *scrum masters*. It is customary to keep these meetings rather short, ranging from 0.5–1 hour. The first three issues discussed are often the same as in the daily stand-up meeting. However, the questions are answered from a team perspective, not for each individual team member:

- What have your team done since the last meeting?
- What is your team planning on doing towards the next meeting?
- What impediments are threatening your team's work, if any?

The next questions to be answered are specific to the Scrum of Scrums meeting and asked to resolve potential issues in order to ease the integration process (Paasivaara, Durasiewicz, & Lassenius, 2008b):

- Have your team placed any impediments in the other teams' way?
- Is your team planning on placing any impediments in other teams' way?

User story: User stories are a common addition to Scrum and creating user stories is a technique often used for fleshing out product backlog items during the sprint planning meeting. Furthermore, according to Mike Cohn (Cohn, 2004), the user story should be written as a description of the desired functionality as seen from a user or customer perspective. The purpose of a user story is to remind the developer(s) working on it, to communicate with the customer or *product owner* in order to further clarify the particular

task. The details behind the story become evident during the conversation with the *product owner*. Roles are typically incorporated into stories which allow them to be written on the form: As a <user>, I want to <goal> so that <benefit>. Furthermore, acceptance tests confirm that the user story was coded in a satisfactory manner. The goal when developing a user story is to make it independent, negotiable, valuable, possible to estimate, sized appropriately and testable, as described by Mike Cohn.

Team velocity: A metric often used when managing Scrum projects is team velocity, which represents how much product backlog effort a team is able to deliver in one sprint. It is an a posteriori value usually developed after the same team has worked together across several sprints and is measured in story points (or hours) per sprint. The team focus in regards to this measure should be; how much value they can add to the business and ***not*** how much code they can produce. During the sprint planning, the team commits to the amount of work they feel they can deliver, however, when the team velocity metric is established this decision is much easier to make as both the *scrum master* and the team are aware of how much the team is capable of producing during a sprint. Normally, the team achieves a fairly stable velocity rather quickly, that is, after 2–3 sprints.

2.3 Project Management in Scrum

This section examines Scrum with a more specific focus on project management and expands on some of the Scrum elements from section 2.2 by providing findings from studies in the literature.

The Scrum-view of project management is analogous to planning a car trip to another city that you have visited many times before. From past experiences, you know how to get there, however, you can't describe in detail which turns and exits you will take at what point in time. Every trip is distinct; however the main parts of the trip are the same e.g. the main roads used, the cities you pass by. There might be some roadwork along the way or an accident that force you to take a minor detour, but you will still reach the destination in the end. Furthermore, you have a rough idea of how long it is going to take, although you can't be precise by the minute.

In Scrum there is no sophisticated use of Gantt diagrams in order to break down every part of the project. Tasks that are to be completed in the far future will not have specific

deadlines. The reason being that, as in a long drive, too much can happen along the way, such as changing requirements, people leaving the project or any unpredictable delays of prerequisite tasks. Thus only rough estimates will be needed when planning the project, while the upcoming sprint will be planned in detail. When a sprint is finished, the deliverables from that sprint will be fully functioning, tested and integrated with all previous work in the project. As a consequence of this fact alone, it is likely that there will be fewer unpleasant surprises along the way that one has to take into consideration when planning a Scrum project.

One of the most valuable elements of Scrum, as reported in studies such as (Paasivaara et al., 2008b), is the daily stand-up meeting. It is a very useful practice when managing projects as much less time is spent in meetings, thus allowing more time to producing quality software. In addition, problems become more evident by informing each other about possible impediments at a daily basis. This makes it possible to handle issues that arise immediately, although not during the meeting, but by setting up a new meeting to discuss the problem at hand if necessary. The stand-up meeting also facilitates effective knowledge sharing as each member of the team make the others aware of what he/she have been working on lately and what will be done the following day. Impediments are also identified earlier and removed quicker after being highlighted during the meeting.

The sprint retrospectives, product and sprint backlog may in many cases contribute to new systematic ways to communicate between the Scrum team and the stakeholders, as highlighted in (Pikkarainen et al., 2008). Furthermore, the increased communication may prove to be vital when attempting to uncover customer expectations and to ensure that the product under development meet the demands of all the stakeholders. The reason being that communication typically plays an important part in any kind of software development project and is regarded as a significant success factor. When considering Scrum, (Pikkarainen et al., 2008) argue that practices such as sprint planning, daily stand-up and open office space have positive effects on internal communication within the development teams. Pikkarainen et al. further argues that an open office space leads to increased informal communication, which in turn contributes in making formal documented communication within the development team increasingly superfluous. An open office space may prove effective in encouraging discussions as people might overhear problems that others are having and offer their expertise thus speeding up the problem solving process. It also

becomes much easier for i.e. the *scrum master* to deliver a message to the whole team since they are all located within in the same area with no walls separating them. Thus the message only needs to be communicated once, which makes project management more effective. (However, an open office space may also be perceived as distracting and frustrating when people are having discussion while others are trying to focus on, say, mind-boggling programming problems.) Further on, it is brought up in (Mann & Maurer, 2005) that sprint planning meetings serve to avoid uncertainty by clarifying for both the developers and customers what is to be developed in the upcoming sprint. Thus it is evident that Scrum places great emphasis on communication, which would entail that Scrum as a methodology for project management is well equipped for success from a theoretical standpoint.

2.3.1 What Scrum does not offer

When considering traditional project management there are several aspects that are not covered by Scrum. Mandatory project deliveries such as a project plan, communication plan, cost benefit analysis, risk tables etc. needs to be handled in the traditional project management fashion. As a consequence, several companies in the industry, which I know of, try to keep the project manager role strictly separated from the *scrum master* role in the projects that they run. Although, in some cases where the same person handles both roles, it is of utmost importance to be fully aware of the differences between the roles in order to separate the appurtenant duties and responsibilities.

A typical problem related to software development is that massive amounts of code builds up as the project progresses and as code keep being added to the product, it becomes more difficult to maintain, extend and debug the code base. Further on, as Scrum is mainly a project management methodology it does not offer guidelines for how to maintain the code base. Consequently, it is very important to complement the Scrum practices with other techniques to ensure code quality. One of these practices is testing - more specifically test-driven development (known from XP). Another is automated testing, which offers many advantages such as making the testing repeatable, helps avoiding mistakes, saves time. Furthermore, by employing continuous integration all code can be compiled, run through automated tests and deployed to a test environment available to the team, the *product owner* and potential users.

Another useful principle that may be beneficial to incorporate in Scrum, or any software development methodology for that matter, is; "Don't live with broken windows" (Hunt & Thomas, 1999), which means that problems with the code should be fixed immediately as this makes the code base easier to maintain. The consequence of not following this principle is that people tend to notice that bad coding practices exist and after a while a common understanding that non-neat coding is accepted, will emerge. This will only make matters worse and the code base will eventually become virtually impossible to maintain and extend (due to bad naming conventions, spaghetti code and workarounds).

2.4 Project Management: Scrum vs. Plan-Driven Development

The linear plan-driven development methods were criticized already from the inception, perhaps surprisingly, also by Royce in his own paper that outlined the thoughts behind the waterfall model (Royce, 1987). The main arguments being that the waterfall process lacked feedback mechanisms from one phase to the next[8]. However, a re-edited version including feedback loops was released, although this version never gained wide acceptance.

Traditional project managers manage projects in regards with budget, schedule and scope, while tracking metrics and variances against planned baselines in order to reduce risk and preserve constraints imposed by time and money (Fernandez & Fernandez, Winter 2008-2009). This role separates itself from a typical *scrum master*, who is more focused on deliverables and business value as well as soft management skills, due to the relatively low emphasis on documentation and the volatile and iteration-based form for planning. Further on, one of the most prominent differences between the two roles is that the project manager's main task is to be responsible for the project, while the *scrum master* should assist the team to deliver a product by removing impediments, instead of strictly following a process. A substantial distinction is also that the *scrum master* directs focus towards team leadership and not control.

In a traditional plan-driven development approach, teams typically consists of people with skills specialized within one area e.g. databases, web services, GUI-programming (Moe & Aurum, 2008). The intuition being that a team of experts within different areas will be

versatile and suitable for most projects, since they would cover such a broad spectrum of skills and still have deep understanding within each field. A consequence of such a team composition, is that each person will be working alone on the module in which that person is a specialist. However, the lack of overlapping skill sets within the team may reduce flexibility and thus decrease internal team autonomy (Moe et al., 2009). In addition, knowledge transfer across the team can be obstructed and make it easy to lose sight of where the project is going. Unless regular meetings are facilitated, in order to maintain communication and build trust, the people in the team might not know what the others are working on. Consequently, a situation where no one will have a complete overview of the progress may occur which, in turn, will result in a lack of understanding of the product.

In contrast, within cross-functional and self-organizing Scrum teams, the team members fill each other in on what they are working on during the daily stand-up meeting. Furthermore, team members often pair up to solve a particular task, thus making them benefit from the synergy effects obtained from such collaboration. Team members will be able to learn from each other and knowledge about the product is more easily spread across the team when new pairs are made up, at a later stage. Due to knowledge being shared and transferred within the team, the consequences of a person being sick or quitting can also be confined. In addition, the overlapping of skills allows for some freedom of choice when delegating the tasks, which makes it possible to hand a particular task to a more motivated individual within the team. However, the team members must be exercise a greater level of commitment as they are required to take on more responsibility through for instace self-organization, than in traditional projects.

Furthermore, in addition to knowledge transfer within the team, the product backlog allows the team members to maintain an overview of the project, since the project goals can be found in the backlog. In contrast, such information may not be explicitly communicated when employing a plan-driven approach. Thus leading to a higher level decision-making process, where only individuals such as the chief architect or management personnel can participate. As a result, making changes can be cumbersome and mistakes are more likely to be made since the project goals and target objectives are not properly communicated.

[8] Sommerville, I. (1996). Software process models. ACM Comput. Surv., 28(1), p.269-271.

Additionally, due to lack of dialogue between the customer, management and the team, problems that need to be discussed might not be discovered before months have passed. However, with Scrum there is, as previously mentioned, much higher emphasis on communication. As argued by Karlström & Runeson (Karlström & Runeson, 2006), practical issues can be identified and resolved earlier in teams following agile principles than in plan-driven projects. Furthermore, it is also argued by (Pikkarainen et al., 2008) that increased informal communication contributes to reduce the need for formal documentation within the team, thus leading to more productive software development than what is achieved through the plan-driven approach. The increased communication is, in part, facilitated by the flat organizational structure, the short iterations and the daily stand-up meetings which support problem identification and resolving to a much larger extent than what is seen in the traditional approach. This is backed up by (Rising & Janoff, 2000) which argues that short time boxed iterations is a key factor for increased communication in teams running agile software development. However, there is also a risk of relying too much on tacit knowledge transfer, either through socialization or externalization, because people might not be willing to share. If such a situation occurs, it will most often be caused by lack of trust or insufficient incentives for knowledge sharing.

To sum up, software development is a complex process that often involves many stake-holders, changing requirements and intricate problems that may change or are not completely understood as well as a very high focus on cost, time/deadlines and quality, but also several other factors. Consequently, it is extremely difficult to carry out all the planning in advance, with the same high focus on details for all stages of the project, which is customary in traditional plan-driven development projects. An alternative approach is to use Scrum and as seen from (Schwaber & Beedle, 2001), Scrum is often adopted in project management situations with rapidly changing environments where it might prove difficult to plan ahead. Consequently, this may indicate that Scrum is a more appropriate develop-ment methodology to employ as software is developed in a very rapidly changing environment. To handle such an environment, Scrum extends incremental software development to what is called “empirical process control”, where feedback loops is the core element (Schwaber & Beedle, 2001).

The table below, table 2.1, seeks to summarize the most important differences between plan-driven development and Scrum based on the discussion in this section and ideas adopted from (Nerur & Balijepally, 2007).

Practices:	**Plan-driven development**	**Scrum**
Management:	Project Manager (*PM*) – command-and-control	*scrum master* (*SM*) – leadership style, collaborative
Team skills:	Team members with specialized skills	Cross-functional teams
Structure:	*PM* leads the team and delegates tasks.	Self-organizing teams whereas the team members flesh out tasks, estimate and choose tasks (not the *SM*).
Mode of communication:	Formal	Informal
Regular meetings:	*PM* usually schedule regular team meetings (lasting 1hour++) to discuss progress in regards to estimates, issues and risk.	Daily stand-up meeting (lasting for 15 min), and also replaces status rapports.
Issue handling:	*PM* compiles and updates an issue log and oversees that the issues are resolved.	*SM* removes impediments holding up the team.
Risk management:	The *PM* is responsible for overseeing the risk analysis and contingency planning as well as the appurtenant document publishing.	Risk is assessed continuously throughout the project e.g. during the sprint planning and the daily stand-up meeting.
Development model and phases:	Project life cycle model (Waterfall, Spiral, RUP or variations of these). Projects consist of a planning phase, then implementation and delivery. The project cycle is guided by tasks or activities.	Evolutionary iterative/sprint-delivery model. Projects are executed iteratively, whereas each iteration/sprint consists of planning, implementation and delivery. The project cycle is guided by items in the product backlog.
Change Management:	*PM* oversees that changes to the original scope are documented, submitted and approved.	During a sprint the whole team must agree before accepting a change. Otherwise changes can be added to the product backlog at any time.
Delivery deadlines:	If a delivery deadline cannot be reached, the *PM* is responsible for negotiating time, quality or cost or some combination of them.	If it appears that the sprint backlog is too big, the additional items will be carried over to the next sprint. The sprint itself is never extended, nor does the *SM* negotiate any project variables.
Demonstration of deliverable:	The only demonstration of the product is during hand-over after the project is finished, unless user-tests or usability tests have been conducted along the way.	The deliverables from each sprint is demonstrated to the *product owner* during the review meeting.
Documentation:	High focus on producing documentation. Often also called document-driven development	Only document the essentials. Relies on internal face-to-face communication in order to share knowledge.
Control:	Process-centered	People-centered

Knowledge Management:	Focus on explicit knowledge. Knowledge is often created through externalization and combination, and shared in an explicit form.	Focus on tacit knowledge. Knowledge is often created through socialization (and internalization), and shared in a tacit form
Customer involvement:	Regarded as valuable	Essential

Table 2.1: Scrum vs. Plan-Driven Development.

2.5 Team Leadership

In any project it is evident that teamwork is important, as projects seldom are carried out by one individual, but rather by one or several teams. Consequently, it is essential to know how to lead the different teams in a project in order for it to be successful. Therefore, one of the main ingredients of project management in agile development can be seen as team leadership. Over the years several frameworks for classifying teamwork behaviors have been suggested (Mathieu & Zaccaro, 2001), although this effort has not succeeded in establishing a scientific consensus regarding the underlying mechanisms of teamwork behavior. In response, to the rather fragmented studies of teamwork and their deficient applicability, (Salas et al., 2005) analyze existing literature, in this field, in an attempt to construct a practically functional framework for understanding teamwork behavior through the "Big Five". In their paper, Salas et al. ground the framework both theoretically and empirically. Furthermore, the appropriateness of using the "Big Five" as a framework, when studying projects to attain a deeper understanding of the challenges companies are facing when implementing agile development processes, is backed up in (Moe & Dingsøyr, 2008). Moe & Dingsøyr make use of the framework to a great extent in their paper and also recommend using it in future work.

According to (Salas et al., 2005), the skill of team leadership is among the „Big Five" in teamwork and can be defined as:

> "The ability to direct and coordinate the activities of other team members, asses team performance, assign tasks, develop team knowledge, skills, and abilities, motivate team members, plan and organize, and establish a positive atmosphere."

Furthermore, according to (Salas et al., 2005), team leadership consist of several behavioral makers:

- Facilitate team problem solving.
- Provide performance expectations and acceptable interaction patterns.
- Synchronize and combine individual team member contributions.
- Seek and evaluate information that affects team functioning.
- Clarify team member roles.
- Engage in preparatory meetings and feedback sessions with the team.

These behavioral makers of team leadership, with the exception of "Clarify team member roles", will serve as my framework when attempting to characterize project management in mature Scrum teams. The reason to why we will exclude the 5th item is that Scrum considers the team as a role in itself. In Scrum, each team is self-organizing and consists of

people with cross-functional skills. As a consequence, the team members' areas of responsibility will vary both within a sprint and across sprints. This means that a particular team member may work as a developer for a couple of days, then as a tester and after that write documentation all within the same sprint.

Each of the selected behavioral makers of team leadership can be explained in more depth based on (Salas et al., 2005):

Team problem solving

Team leadership affects team effectiveness by facilitating team problem solving through cognitive processes such as shared mental models[9], coordination[10] and the team's collective motivation[11] and behaviors which may be fueled by performance expectations (see para-

[9] Shared mental models is an organizing knowledge structure of the relationships among the task the team is engaged in and how the team members will interact.

[10] Through coordination the team members can cooperate by anticipating and predicting each other's needs through common understandings of the environment and expectations of performance.

[11] Collective motivation means the that there exists a common understanding, eagerness and feeling of responsibility within the team to successfully complete the tasks at hand.

graph below). The team leader has a role in the creation, maintenance and accuracy of the team's shared mental model.

Performance expectations and interaction patterns

It is the team leader's responsibility to establish behavioral and performance expectations as well as tracking the abilities and skill deficiencies of each team member. The team leader should set expectations and promote acceptable interaction patterns that encourage information transfer and knowledge sharing. This may create a team climate that supports behaviors such as mutual performance monitoring[12], backup behavior[13] and adaptability[14]. If task- and team-based norms for interaction are established, it will allow team members to affect each other so that team expectations and norms are conformed to. This may ground self-organization as a norm in itself within the team.

Synchronize team member contributions

Team leaders facilitate team leadership by synchronizing and combining the individual contributions of each team member and by making sure that the individuals on the team recognize their independence as well as the benefits from the synergy effects gained by working together and joining forces. By reinforcing the needed processes within the team this may ensure that the desired behaviors occur and affect the manifestation of the trained skills that reside within the team members.

Information affecting team functioning

The team leader has often accurate and comprehensive information regarding the teams' situation, and is thus often in the best position to establish and maintain an accurate shared understanding of the team objectives, the team constraints and the resources available to the team. By providing the team members with such enriched information they may devel-

[12] Mutual performance monitoring is the ability to develop common understandings of the team environment and apply appropriate task strategies to accurately monitor teammate performance.

[13] Backup behavior portrays the concept of redundancy in teams. It is an (imporatant) mechanism that affects the team's ability to adapt to changes in the team environment. According to Marks (Mathieu & Zaccaro, 2001) there are three ways of providing backup behavior: providing a teammate verbal feedback or coaching, assisting a teammate behaviorally in carrying out a task or to complete a task for the team member when an overload is detected.

op more similar mental models which in turn allow them to align which brings them closer to being self-organized. Based on this, they may make informed decisions.

The team leader also facilitates team leadership by monitoring the internal and external environment to the team. This information may be used to facilitate team adaptability and to prepare the team for changes in the environment. Furthermore, information regarding external factors may assist in coordinating suitable team behaviors and interactions as well as indicate areas for skill development.

Meetings and feedback sessions

By engaging in preparatory meetings with the team, the members may synchronize their effort with regards to future work. However, this requires both team orientation[15] and shared mental models in order to be successful. Based on previously recorded team performance, new and improved estimates can be established and informed planning sessions carried out.

When the team completes an iteration, team members progress into a transition phase in which they reevaluate their performance, provide and receive feedback, and make adjustments to their strategies as needed. By engaging in feedback sessions, team member feedback can lead to individuals becoming more aware of their effectiveness and the quality of their work. During such sessions, teams will require team leadership to provide additional guidance and feedback. Members' team orientation will play a role in receiving and using performance feedback from the team leader and other team members. Information gathered through mutual performance monitoring, and expressed through feedback and exercising backup behavior, affects team performance by allowing the team to amend identified errors or lapses which, in turn, boosts the sum of individual achievements and increases the synergy of teamwork. A shared understanding of the teamwork also acts as a foundation for the effectiveness of performance monitoring and feedback, as a lack thereof makes any feedback that could potentially be given inconsequential.

---

[14] Adaptability is the ability to adjust strategies based on information gathered from the environment through the use of backup behavior and reallocation of intrateam resources in response to changing conditions.

[15] Team orientation is the propensity to take other's behavior into account during group interaction and belief in the importance of team goals over individual members' goals.

3 Research Methods and Design

The research methods and approaches employed to determine what characterizes project management in mature Scrum teams were a literature study and an empirical multi case study. The motivation behind the choice of methods and the details of how the data collection was carried out are presented in section 3.1, whereas section 3.1.1 is devoted to the literature study and section 3.1.2 to the empirical multi case study. Furthermore, as the data obtained using both research methods was analysed more or less in the same manner, a common description of the data analysis is given in section 3.2.

3.1 Data Collection

Information was gathered through a literature study in order to get an overview of available theory and existing empirical studies. In order to examine how Scrum is practiced in the industry today, an empirical multi case study was performed where a range of projects from four different companies were selected and key personnel interviewed.

3.1.1 Literature study

In order to get an in-depth understanding of project management in Scrum, I decided to perform a thorough literature study. Through this examination it was also possible to identify what kind of work that had been done previously, thus being able to focus on areas that have not received enough or the proper attention earlier. Ideas from different papers also inspired to further work and elements from the various fields could be combined in new ways. By gathering extensive amounts of existing relevant information about the domain, the necessary background information was in place, before embarking on an empirical multi case study. This directed more emphasis to the practices identified to be of importance through the literature study and the common problems experienced in the Scrum projects, which allowed me to be better prepared for the empirical multi case study.

Selection Process

When searching for Scrum studies, I used various sources of information. In this process I searched through different databases and when I identified a resource that satisfied my

needs, its reference list was investigated in order to obtain new sources of potential value. The databases I scanned are listed below:

- IEEE.org Xplore (http://ieeexplore.ieee.org/)
- Elsevier (http://www.elsevier.com)
- Science Direct (http://www.sciencedirect.com/)
- Portal (the ACM Digital Libraries) (http://portal.acm.org/)
- ISI Web of Knowledge (http://apps.isiknowledge.com)
- Springer Verlag (http://www.springerlink.com/)
- Scientific Commons (http://en.scientificcommons.org)

In order to know what to look for when selecting quality results relevant to our research, there was a need to establish certain criteria. The criteria used in the selection process were that the article had to be:

- **A Scrum study**, meaning that the main parts of the article must be about the Scrum principles and project management issues.
- **An empirical research study** from a company in the industry, which means that studies performed on student teams are not considered relevant.
- **A scientific article**, that is, not a "lessons learned"-report. In order to be classified as a scientific article there must be a research question, a description of the method used to solve the problem at hand and a section giving the research findings.

During the search process, the title of all the results was investigated. This can be considered as the first phase of the search. Whenever a retrieved article seemed to be of interest, judging from the title, it was further examined by inspecting its abstract. In this second phase of the selection process, most of the resources that only fulfilled some of the criteria were discarded. However, in some cases it required me to read parts of the article in order to determine if all the criteria were fulfilled. The results that were found to be most satisfactory are listed in the table below.

Below, in table 3.1 the different keywords used when searching the various databases are given along with the available results returned, the number of articles selected and the

name of the selected article. Some of the selected results were returned by several databases, these are only listed under the first database from which they were returned.

Keywords used:	# Total results:	# selected in phase 1:	# selected in phase 2:	Name of article:
scrum AND agile				
- IEEE.org Xplore	83	14	4	Scrum in a Multiproject… Dist. Agile Dev using Scrum… Agile methods in European.. Understanding Decision-Making…
- Elsevier	0	0	0	-- None --
- Science Direct	36	3	0	-- None --
- Portal - ACM Digital Lib.	152	3	1	Scrum Down – A SW engineer…
- ISI Web of Knowledge	16	3	2	The impact of agile practices on com. Customizing agile methods…
- Springer Verlag *limit the search to English results only.*	199	3	3	Scrum Implementation using Kotter's.. The impact of Methods & Tecnnique.. Developing SW with Scrum in...
- Scientific Commons	73	5	2	Scrum med XP-relaterade.. Agil videnledelse med Scrum..
agile AND software *yield too many results!*				
- IEEE.org Xplore	823!	…	…	…
- Elsevier	20	0	0	-- None --
- Science Direct	2518!	…	…	…
- Portal - ACM Digital Lib. *limit search results from 2000-2009*	1726!	…	…	…
- ISI Web of Knowledge *limit search results from 2000-2009*	339	2	0	-- None --
- Springer Verlag *limit search to English results from 2000-2009*	3440!	…	…	…
- Scientific Commons *limit search to English results only*	984	2	2	Lean and Agile Software Dev... R&D Process Optimization..
scrum AND software				
- IEEE.org Xplore	74	2	0	-- None --
- Elsevier	0	0	0	-- None --
- Science Direct *excluding irrelevant journal/book titles and releases before 2000*	56	0	0	-- None --
- Portal - ACM Digital Lib.	163	0	0	-- None --

- ISI Web of Knowledge	12	1	0	-- None --
- Springer Verlag *limit search to English results from 2000-2009*	214	2	1	Scrum and Team Effective: Theory..
- Scientific Commons *limit search to English*	36	0	0	-- None --
Total Findings	10964	40	15	

Table 3.1: The results obtained when searching the databases.

The main articles found amount to 15 studies, as seen from table 3.1. None of them are published prior to 2004. After performing the search process, the bibliographic lists of the selected articles were inspected in order to obtain new relevant articles. Nine articles fulfilling the criteria, established for the selection process, were obtained, which amounts to 23 articles in total (as one of the articles is just a compressed version of another). All together, there is 1 action research study, 12 case studies, 3 ethnographic studies, 2 surveys, 2 holistic studies and 3 other kinds of study. The articles in the category "other", does not fall into any of the previously defined categories. Furthermore, according to (Dingsøyr et al., 2008), the distribution between journal and conference publications can indicate the maturity of the field and of subfields. When considering the selected articles, there are 6 journals, 11 conference publications, 1 book chapter and 2 magazine articles while the rest is unpublished work such as white papers, PhD and master thesis, thus indicating that the Scrum field is maturing, although it is still young and much research remains to be carried out. When looking more specifically at the categories of studies within the agile field, that were selected, we find that there are 14 Scrum studies, 1 XP study[16], 4 studies of both Scrum and XP as well as 4 studies of agile in general.

At this point, however, the list of articles was too diverse in terms of quality. In order to further refine the list of articles, I decided to only focus on studies with a certain depth to focus on the more comprehensive and high quality studies. Consequently, only studies of 8 pages or more were accepted. However, it may be argued that some thorough and highly distinctive articles may be lost due to this requirement. Although, after evaluating the

[16] Included because it describes elements found in Scrum, even though it is a case study of XP teams.

articles found to be of less than 8 pages, I decided that too many articles would have been included (if this requirement was to be dropped) which would disperse attention without adding sufficiently new information concerning project management in Scrum. Furthermore, I decided to remove master thesis and PhD dissertations from the selected resources since these in most cases contains much more than a typical empirical Scrum study and includes great deals of general information about the field that are not applicable to my research objectives. In order to maintain focus, I also removed several studies with predominance on XP or lessons learned.

Table 3.2, below, indicates the name of each study, where it is published (if applicable) and whether it is included in the list of the selected articles. If an article is not included, the reason to this decision is given in the far right column. Further on, the table lists the included articles first in alphabetically order, while the other articles are grouped by the criterion from which they were removed.

No	Studies:	Source:	Included	Removed by criterion
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction (10 pages), (Mann & Maurer, 2005)*	Proceedings of the Agile Development Conference '05	Included	
2	*Customising agile methods to software practices at Intel Shannon (14 pages), (Fitzgerald et al., 2006)*	European Journal of Information Systems 2006	Included	
3	*Developing Software with Scrum in a small Cross-Organizational Project (11 pages), (Dingsøyr, Hanssen, Dybå, Anker, & Nygaard, 2006)*	EuroSPI 2006	Included	
4	*Scrum and Team Effectiveness: Theory and Practice (10 pages), (Moe & Dingsøyr, 2008)*	XP 2008	Included	
5	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges (12 pages), (Marchenko & Abrahamsson, 2008)*	Agile 2008 Conference	Included	
6	*The impact of agile practices on communication in software development (35 pages), (Pikkarainen et al., 2008)*	Springer Science + Business Media	Included	
7	*Using Scrum in a Globally Distributed Project: A Case Study (18 pages), (Paasivaara et al., 2008b)*	John Wiley & Sons, Ltd	Included	
8	*Understanding Decision-Making in Agile Software Development: a Case-Study (8 pages), (Moe & Aurum, 2008)*	34th Euromicro Conference Software Engineering and Advanced Applications	Included	

9	*Understanding Self-organizing Teams in Agile Software Development (10 pages), (Moe et al., 2008)*	19th Australian Conference on Software Engineering	Included	
10	*Understanding Shared Leadership in Agile Development: A Case Study (10 pages), (Moe et al., 2009)*	Proceedings of the 42nd Hawaii International Conference on Systems Sciences - 2009	Included	
11	*Agile methods in European embedded software development organizations: a survey on the actual use and usefulness of Extreme Programming and Scrum (7 pages), (Salo & Abrahamsson, 2008)*	IET Software (journal)	---	Too short (not an empirical case study)
12	*Theoretical reflections on agile development methodologies (5 pages), (Nerur & Balijepally, 2007)*	Communications of the ACM	---	Too short (A bit off topic not an empirical case study not a scientific article)
13	*Agile Project Management: Steering from the Edges (5 pages), (Augustine, Payne, Sencindiver, & Woodcock, 2005)*	Communications of the ACM	---	Too short (not a scientific article)
14	*Preliminary Investigation of Stand-up Meetings in Agile Methods (3 pages), (Hasnain & Hall, 2008)*	IAENG International Journal of Computer Science	---	Too short (not a scientific article)
15	*Scrum Down: A Software Engineer and a Sociologist Explore the Implementation of an Agile Method (4 pages), (Bates & Yates, 2008)*	CHASE '08	---	Too short (not a scientific article)
16	*Agil videndelse med Scrum – en undersøgelse af dokumentationens rolle i systemudvikling (154 pages), (Sørensen et al., 2008)*	Not published	---	Master thesis / PhD thesis
17	*Lean and Agile Software Development: A Case Study (90 pages), (Murray, 2008)*	Not published	---	Master thesis / PhD thesis
18	*R&D Process Optimization for a Customer and Order Management System (76 pages), (Halkola, 2008)*	Not published	---	Master thesis / PhD thesis
19	*Scrum med XP-relaterade tekniker – Införandet av Scrum och dess påverknad på systemutvecklargruppen (181 pages), (Forsberg & Lindström, 2008)*	Not published	---	Master thesis / PhD thesis
20	*The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects (15 pages), (Parsons et al., 2007)*	Boston: Springer	---	XP Study (mainly) (not an empirical case study)
21	*Agile Project Development at Intel: A Scrum Odyssey (14 pages), (Elwer, 2008)* 2008 Danube Case Study: Intel Corporation	White paper from Danube Technologies	---	Lessons learned not a scientific article
22	*Scrum Implementation Using Kotter's Change Model (11 pages), (Hayes & Richardson, 2008)*	XP 2008	---	A bit off topic not a scientific article
23	*An Iterative Improvement Process for Agile Software*	John Wiley & Sons,	---	A bit off topic

	Development (20 pages), (Salo & Abrahamsson, 2006)	Ltd.		
24	*Distributed Agile Development: Using Scrum in a Large Project (9 pages) (A short version of no.7), (Paasivaara, Durasiewicz, & Lassenius, 2008a)*	2008 IEEE International Conference on Global Software Engineering	---	Not included because a more comprehensive version exists

Table 3.2: Refinement of articles

As portrayed by table 3.2, ten studies were included in the list of the selected articles. Among these studies, three (study-4, -8 and -9) reported from the same project. However, different aspects of the project were studied in each case.

3.1.2 Empirical multi case study

As highlighted in (Dybå & Dingsøyr, 2008), there is a lack of Scrum studies with adequate quality, especially in mature teams. Based on this result, I decided to investigate what characterizes effective project management in mature Scrum teams developing software products. As I intend to examine project management in mature Scrum teams, which is a contemporary phenomenon in a real-life context, the descriptive and exploratory nature of an empirical case study seemed to be most appropriate approach (Benbasat, Goldstein, & Mead, 1987; Yin, 2002). Moreover, a case study is suitable in research situations where research data are collected through observation (e.g. taking pictures of the team setting and the task board) in an unmodified setting (Yin, 2002). By conducting interviews, and in some cases being on-site observing for a limited time, it was possible to record a more genuine representation of the daily work and capture tacit knowledge from the employees. It also allowed for a more objective and non-biased information gathering than what can be obtained through for instance an internal lessons learned report.

An often used argument against case studies is that it does not offer inductive inferences, whereas a common argument is that the information gathered is in many cases too tied up to the particular circumstances under which the study was performed. However, according to (Yin, 2002), it is possible to make theoretical generalizations even from a single case study. Further on, when performing a quantitative study the sample size must be sufficient and taken from a population that is representable in order to say something about the whole population in general. A qualitative study like this, on the other hand, offers more in-depth knowledge through the results presented and thus more valuable to understand the underlying mechanisms at hand after a thorough analysis (Yin, 2002). The extensive results

produced by this study may later be utilized to create hypotheses which in turn can be tested in a quantitative manner through i.e. a survey.

The empirical multi case study was conducted as a series of informal semi-structured interviews[17] with development team members and *scrum masters*, in the studied organizations, and in some cases notes and pictures were taken on-site. The interviews were performed, in the form of a dialogue, using the interview instrument (see Appendix A) as a guide of areas for discussion. Issues that seemed to be of interest were followed up in order to retrieve more information and to provide clarity. As a result the interview instrument was slightly modified as the interviews progressed. Adaptations were primarily made with the purpose of gaining further information about statements made by previous interviewees. All the interviews were recorded as audio files, and then fully transcribed in order to capture everything during the interview, thus being better positioned to identify interesting patterns and themes.

In total, seven interviews where conducted, which reported from six different projects carried out by four distinct companies. The interviewees were, in some cases, chosen based on convenience, as I have previous work experience with four of the interviewees. Furthermore, as I had knowledge of some of the interviewees beforehand, I was also able to handpick individuals that I knew had been working in mature Scrum teams. The rest of the interviewees where recommended by the others interviewed based on their prior experience with Scrum.

In the first company, three semi-structured interviews were conducted with one developer and two *scrum masters*. All of them were ‘mature’ as the developer had successfully taken part of producing software using Scrum for 2 years, while the *scrum masters* had been working with Scrum for 2.5 years, which is since the company introduced Scrum. In the second company, a semi-structured interview was conducted with a chief architect having prior experience as both a *scrum master* and *scrum master* training instructor. This individual had been using Scrum for over four years while working for two different companies and had a broad experience with agile development and appurtenant problems. Furthermore, this individual is a prominent character in the agile environment, thus being

able to speak more generally about the field. However, the second company itself had only been officially employing Scrum for about a year, while the informant reported from a project he had been a part of for about two months. Further on, two semi-structured interviews were conducted in the third company whereas both interviewees worked on different projects as *scrum masters*. One of the *scrum masters* had been certified Scrum Practitioner after being instated as a *scrum master* for more than one year in the organization. This *scrum master* reported from a project, which he initiated, that lasted for 1.5 years. The other *scrum master* reported from two projects. The first project was initiated by him and lasted for 1.5 years, while the second had been running for three years when he joined and at the time of the interview he had been a part of this project for five months. The company itself has been officially using Scrum for 1.5 years. In the last company, a semi-structured interview was conducted with a developer who had been working with Scrum for almost two years and reported from a project whereas he was involved for five months.

3.2 Data Analysis

The findings from both the literature study and the empirical multi case study was examined, analyzed and discussed utilizing the team leadership framework, given in section 2.5, as a lens. During the analysis of the articles found through the literature study, the various project management practices employed were identified and categorized according to the framework. Furthermore, the studies were compared in order to discover patterns, common practices and typical problems faced. This allowed a foundation to be established, which was used when embarking on the empirical study as some areas of focus was already defined through the literature study. It also gave insight to what the focus in terms of team leadership had been previously and which made it possible to evaluate to what degree this focus has shifted over time. Furthermore, by examining the findings from the interviews from a team leadership perspective it allowed me to better be able to document how these practices were actually performed in the field, based on the particular companies investigated.

---

[17] Robson, C. (2002). Real World Research. Oxford: Blackwell

It should be noted that practices relevant to the behavioral makers of team leadership were not reported from all projects, thus only findings relevant to the framework are presented in section 4.2–4.6 and 5.2–5.6. Furthermore, during the analysis it became apparent that it was not trivial to place aspects of Scrum, indentified from the findings, in regards with the team leadership framework defined by (Salas et al., 2005). The reason being, that several of the Scrum aspects identified can be seen as being related to more than one of the behavioral makers of team leadership. Consequently, the findings are placed under the behavioral maker which I determined it to be most strongly connected to and an argumentation for its placement is provided where I deemed it necessary.

4 Literature Study Results

In this chapter I present the findings from the literature study. The selected studies are categorized according to project size and the projects are characterized in regards with degree of distribution and customer interaction. Further on, I attempt to determine the depth and trustworthiness of the studies. Then, I perform a comprehensive analysis of the selected studies through the lens of team leadership as it is defined by the theoretically and empirically grounded "Big Five" framework found in (Salas et al., 2005). The behavioral makers of team leadership that I will use are;

- ❖ Facilitate team problem solving
- ❖ Provide performance expectations and acceptable interaction patterns
- ❖ Synchronize and combine individual team member contributions
- ❖ Seek and evaluate information that affects team functioning
- ❖ Engage in preparatory meetings and feedback sessions with the team

This analysis will provide an overview of the available literature on project management aspects in (mature) Scrum teams.

4.1 Categorization of Studies

In the first part of this chapter, the selected studies are categorized according to project size, degree of project distribution and customer interaction. This is carried out in order to determine if the literature can provide information regarding how project management is adapted according to the project's extent. Additionally, the data collection and study method are examined to characterize the depth and validity of each study.

4.1.1 Studies according to project size

The studies are characterized according to project size, because there can be a huge difference in how to lead a small project compared to managing a large project. By placing emphasis on the project size as a dimension for differentiation, we aim to discover changes in project management across the studies. Table 3.3, below, characterizes the selected studies according to project size into the categories; small (0–1year), medium (1–1.5years)

and large (1.5years→) project. When attempting to categorize the studies according to project size, the main emphasize was placed on project length while team size has been used as an adjusting factor. It may be argued that a short-term project consisting of a massive team would lead to more complex project management challenges than having a small team work for a long period of time. However, a long-term project most often involves several stakeholders, numerous teams, coordination issues as well as other challenges, thus by trying to adjust my categorization by inspecting the team size as well, the project size categorization is deemed sufficient in my opinion.

Another issue is whether to look at actual project length or the period of time the researchers gathered information used for the study. We chose to focus on the actual length of the project, where this was possible, because the project management measures taken in each case are decided based on the actual project length and not based on, say, the time period the researchers observed daily stand-up meeting. However, the actual project length is not given for all studies, thus the "research intervention period" is used in these situations instead.

No	Studies:	Project size:
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction*	Large
2	*Customising agile methods to software practices at Intel Shannon*	Large
3	*Using Scrum in a Globally Distributed Project: A Case Study*	Large
4	*The Impact of Agile Practices on Communication in Software Development*	Medium
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study 6 and 9 (project North)*	Medium
6	*Understanding Self-organizing Teams in Agile Software Development* *(same project as in study-5)*	Medium
7	*Developing Software with Scrum in a small Cross-Organizational Project*	Small
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	Small
9	*Understanding Decision-Making in Agile Software Development:* *a Case-study (only considering Project South)*	Small
10	*Understanding Shared Leadership in Agile Development: A Case Study*	Small

Table 4.1: The size of the projects in the selected studies.

As seen from table 4.1, the selected studies report fairly evenly from large, medium and small projects. However, a slightly larger share of small projects is reported.

In the following, the selected studies are grouped by category starting with the largest projects. For each category, the studies are characterized by giving the details of the project size for each study as well as to what degree the reported projects were distributed. Furthermore, the customer interaction in the project is also addressed. The reason why these aspects were examined was to attain an overview of the comprehensiveness and the external factors affecting the projects reported in the selected studies. By inspecting how these aspects change according to project size, we may be able to identify how project management is adapted according to the project size. The results of this examination can be found below in table 4.2–4.4, whereas studies (study-5, -6 and -9 – project North) reporting from the same project are categorized according to the time period each particular aspect of the project was observed and according to the team size involved. Abbreviations used in table 4.2–4.4 are: *product owner (PO), scrum master (SM), project manager* (*PM*).

The large projects

No	Studies:	Project size:	Distributed project:	Customer interaction:
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction*	Length: 2 years Team size: 4-7 Sprints: 4-8 weeks	Not distributed	Co-located in the same building with the customer. All developers on the same floor while the primary customer partially on the same floor and the floor below. The customer participated in sprint planning, daily stand-up and review meetings.
2	*Customising agile methods to software practices at Intel Shannon*	Project IXP2XX: Length: 18 months Teams: 4 in total 15 engineers Sprints: 20 working days Project IXP4XX: Length: 24 months Teams: 5 in total 30 engineers Sprints: 20 working days	Distributed: 4 teams across 3 sites 5 teams across 2 sites	External customer. On-site customers are not available. In the early conceptual stages the projects did not have specific customers. The product marketing group acted as a customer proxy, prioritizing features based on potential revenue
3	*Using Scrum in a Globally Distributed Project: A Case Study*	Length: 1.5 years Teams: 7 Team size: 2-9 Sprints: 4 weeks	Distributed: Norway & Malaysia 20 people in each country	Internal *PO.* All customers have access to Jira (issue-handling system) were they report identified bugs. *PO* picks issues from Jira for the product backlog. *PO* attended several meetings.

Table 4.2: Details describing the large projects.

When considering the large projects, all of them can be considered to involve mature Scrum teams as they lasted for 1.5 years or more. Even though the project in study-3 reported to have rather unstable teams, people were mostly swapped between teams, thus they were still gaining Scrum experience through working on the same project. Furthermore, high customer involvement and interaction were found to be an important part of project management. Regular meetings were conducted and the customer or the *product owner* normally attended daily stand-up meetings, sprint planning and review meetings, as reported in study-1 and -3. Further on, it was reported in study-3 that the customers had access to the "issue-handling" tool Jira (see table 4.2), which made it possible for them to add items to the product backlog, but no way to ensure that these were highly prioritized without explicitly contacting the e.g. *product owner*. This may indicate that a high degree of customer interaction through regular meetings is needed to successfully manage large projects. However, as little information is provided in study-2 regarding customer interactions, it is hard to say if this was a common practice in this case as well, thus making it difficult to draw any conclusions or inferences for the general case.

The medium projects

No	Studies:	Project size:	Distributed project:	Customer interaction:
4	*The Impact of Agile Practices on Communication in Software Development*	F-Secure Corporation Project 1: Length: 18 months Team size: 6 Sprints: 4 weeks Project 2: Length: 4 months Team size: 7 Sprints: 1-2 weeks	In both projects: the testing team was located separately	Internal *PO.* On-site customer practices NOT used.All customers could not participate in sprint planning, therefore the *PO* was key person regularly communicating requirements, thus contributing customer input to development team.Customers and *PO* defined work tasks in sprint planning before developers started estimating. Proj 2: E-mailed external stakeholders about project status each working day.
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 (Project North) and -9*	Length: 4000hours Team size: 6 & 1 *SM* 1 *PO* Sprints: 4 weeks	*PO* in another city	*External PO. PO* acted as a representative for the customer who was the local government of a Norwegian city.
6	*Understanding Self-organizing Teams in Agile Software Development* same project as in study-5	Length: 7 months ethnographic study see study-5	see study-5	see study-5

Table 4.3: Details describing the medium projects.

From table 4.3, it is apparent that only project 1 from study-4 can be considered to involve mature Scrum teams as it lasted for 1.5 years. Further on, the medium-sized projects have a low degree of project distribution. However, a *scrum master,* developer or *product owner* may be located in another city without major complications, as reported in study-5, where the project had an external *product owner* who attended daily stand-up meetings over telephone. Although this situation may lead to less customer involvement caused by the distance, as reported in study-5 where the *product owner* was not able to attend all the retrospective meetings. However, regular meetings are conducted in the medium projects, but with lower customer involvement than what were reported for the larger projects. Furthermore, the *product owner* sometimes act as a strong link between the developer team and the customer even though he is internal to the organization, as seen in study-4.

The small projects

No	Studies:	Project size:	Distributed project:	Customer interaction:
7	*Developing Software with Scrum in a small Cross-Organizational Project*	28.april 2005-2006 Length: 8 months Team size: 1 dev. & 1 *PM* from SAM and 2 dev. from Avinor where 1 also was general *PM* Sprints: 10 work-days	contractor and customer were *not* co-located, although the developer from SAM stayed at the customer's site for shorter periods	Applied Mathematics (SAM), Software Process Improvement at SINTEF ICT. SAM was working alone for the first two sprints, but the customer (Avinor) participated in the development team as well from sprint three. Lead to a customer-provider rel.ship between the orgs. Reported on good dialogue and cooperation with customers.
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	April-December 2007 Length: 8 months Teams: 3 Team size: 5-7 Sprints: 6 weeks at most	Not distributed – majority of people were located closely in two-person offices.	At first the PO sometimes asked concrete team members to do critical tasks without consulting the team. During the sprint review the team demonstrated what they accomplished during the sprint to the *PO* and optionally to the customers.
9	*Understanding Decision-Making in Agile Software Development: a Case-study* *Only listing project South details.*	Length: N/A Scrum introduced 1 year after project started Team size: 6 dev. in core team Sprints: 3 weeks	Not distributed	The team got constant feedback from the market through the customer during the daily stand-up meeting, thus it was problematic to protect the sprint backlog against new prioritizing during the sprint.
10	*Understanding Shared Leadership in Agile Development: A Case Study*	(studied for) 1 year Length: 3000 hours Team size: small SW dev. department with 16 dev. Sprints: 1-6 weeks	Not distributed	Little information concerning the customer. The *PO* is selected by the *SM*, the customer and by the management. All stakeholders involved in sprint planning meetings

Table 4.4: Details describing the small projects.

As seen from table 4.4, none of the projects involved mature Scrum teams as they all lasted less than 1.5 years and were typically among the first Scrum projects carried out by the organization. Further on, none of the projects categorized as small were distributed. In a few of the reported projects, it appears to be unclear whether the *product owner* was external or internal, thus providing less information regarding how strong the connection between the team and the customer really was. Although, in some cases this was addressed in more detail; customer involvement was facilitated by inviting all stakeholders to participate in meetings, e.g. the sprint planning meetings, as reported in study-10, while in study-7 the customer participated in the development team as well from sprint three and it was reported on good dialogue and cooperation with the customer.

In some cases less intervention from the *product owner* and the *scrum master* is desired by the team, as described in study-8. Moreover, a high degree of customer involvement may lead to new requirements being added to the sprint backlog during the sprint, thus leading to scope creep (increased scope), a situation described as problematic in study-9 whereas the team received constant feedback from the customers during the daily stand-up meeting. However, customer feedback may very often improve the product and lead to increased customer satisfaction. Customer feedback and involvement may also be achieved, perhaps especially in small projects involving few stakeholders, by having one or more developers co-located with the customer for periods of time, thus increasing communication and cooperation, as described in study-7.

4.1.2 Studies according to depth and validity

When considering the different methods used to investigate and examine what is practiced in the industry, I decided that the "depth" (or how thoroughly they were conducted) of each study is determined by the level of insight it provides (regarding what is practiced in the industry). In order to determine the depth and validity, I examined the data collection type and the study method. The categories; observation, interview, document analysis and questionnaire were established in order to settle on how the data were collected. Moreover, a study may fall under one or more of these categories. In addition, the study methods defined were; multiple-case, single-case, action research and survey. A study can only be found to match one of these categories as they are mutual exclusive. The results of this analysis can be found below, in table 4.5.

No	Studies:	Data collection:	Study method:
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction*	- observation (2 days a week from June 2004) - document analysis (office hour recordings, that is, hours spent at work except for lunch) - questionnaire	Single-case: 2-year longitudinal industrial case study
2	*Customising agile methods to software practices at Intel Shannon*	- interviews (12) - e-mail survey - observation (post-Scrum workshops)	Single-case: 2-year longitudinal case study
3	*Using Scrum in a Globally Distributed Project: A Case Study*	- interviews (7)	Single-case: large distributed project
4	*The Impact of Agile Practices on Communication in Software Development*	- observation - interviews - document analysis (product & sprint backlog, paper worksheet from 17 project workshops, project tasks from the walls, 54 daily status reports)	Multiple-case: multiple-project for 1-year in F-Secure Corp.
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 (Project North) and -9*	- observation (60) (once or twice a week) - interviews	Single-case holistic study (done in context of a larger action research program)
6	*Understanding Self-organizing Teams in Agile Software Development* same project as in study 5	- observation (45) (May 2006 - January 2007, once or twice a week)	Single-case holistic study: 7 months (done in context of a larger action research program)
7	*Developing Software with Scrum in a small Cross-Organizational Project*	- interviews (3) - document analysis (backlog and other doc.)	Action research
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	- observation (daily at first, then daily to weekly 18,075 words diary)	Multiple-case: 8 months, multi-team, multi-project, industrial longitudinal study
9	*Understanding Decision-Making in Agile Software Development: a Case-study only about Project South*	- observation - interviews	Multiple-case holistic study: two projects (North & South)
10	*Understanding Shared Leadership in Agile Development: A Case Study*	- observation (72) (April 2007 – January 2008, one to five times a week) - interviews (3) - document analysis (product and sprint backlog, burndown charts, recorded data from MS Team System and index cards from the wall)	Single-case holistic study: (done in context of a larger action research program)

Table 4.5: Analysis of the depth and validity of the selected studies.

From table 4.5 it is apparent that six of the ten selected studies are single-case studies, thus it may be argued that this represents a threat to the validity of the results presented in these studies.

As we are examining the available literature for findings describing aspects of project management through a team leadership lens in (mature) Scrum teams, it is reasonable, in my opinion, to inspect the method used for data collection in the articles found to be most relevant to my research. It is evident from table 4.5 that 70% of the studies involve multiple forms of data collection, meaning that several aspects of the particular project(s) have been captured. The most common forms of data collection are observation and interviews, semi-structured interviews to be specific. As these are the most frequently used methods for data collection, in the studies considered most relevant to my research, it may indicate that conducting interviews and observations are examination schemes needed to study project management.

A possible threat against validity in this respect is that in some cases only a small number of interviews have been conducted (study-3,-7,-10), while the quantity is not stated for all studies employing this form of data collection (study-4,-5,-9). Further on, the number of observations carried out will also affect a study's validity, although this parameter is not given in every study either. However, in the studies where this information is provided, observations seem to have been conducted 1–2 times a week, which is considered satisfactory in regards to frequency. Nonetheless, the time frame and when each observation was carried out, still remains as important factors when considering the value to assign the contribution of every observation for the particular studies. In regards to these aspects several studies report that the observations were conducted during the time period of the daily stand-up (study-5,-6,-8,-9,-10), planning (study-5,-6,-8,-10), review (study-5,-6,-8,-10) and retrospective meetings (study-4,-5,-6,-8,-10). This is deemed satisfactory in my opinion, as the major decisions and discussion take place during these meetings, thus making them valuable for extracting useful information.

Further on, three studies are longitudinal, meaning that a considerable amount of time has been devoted to comprehend the conditions in the examined project(s). Consequently, based on the above discussion, it is my opinion that these observations signify the investment of thorough and solid work in regards to the selected studies, thus indicating that they are highly valid.

4.1.3 Studies according to team leadership based on the "Big Five"

Abbreviations used in the table 4.6–4.11 are: *product owner (PO), scrum master (SM), project manager* (*PM*).

No	Studies:	Team problem solving	Performance expectations & interaction patterns	Synchronize team member contributions	Info. affecting team functioning	Meetings and feedback sessions
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction:*	Sprint planning meetings with customers, *SM* did preparatory planning with customer identifying ideas to help developers estimate	Pair programming	Several findings identified	No mention of burndown charts	Several findings identified
2	*Customising agile methods to software practices at Intel Shannon:*	Several findings identified	Several findings identified	Several findings identified	Several findings identified	Several findings identified
3	*Using Scrum in a Globally Distributed Project: A Case Study:*	Sprint planning: distr. meeting, local meeting in Norway and Malaysia	---	Daily stand-up 15min with *PO* and 1 meeting room on both sites with teleconference connection & Web cameras	Separate backlog for each team in Jira, all team members can access, all *PO*s can add issues to maintenance backlog	Several findings identified
4	*The Impact of Agile Practices on Communication in Software Development:*	Several findings identified	Several findings identified	daily stand-up meetings at the end of each day,	Several findings identified	Several findings identified
5	*Scrum and Team Effectiveness: Theory and Practice reported info also taken from: study-6 and -9 (Project North)*	Several findings identified	Several findings identified	Several findings identified	Several findings identified	Several findings identified
6	*Understanding Self-organizing Teams in Agile*	same project as in study-5,	same project as in study-5,	same project as in study-5,	same project as in study-5,	same project as in study-5,

	Software Development	see study-5	see study-5	see study-5	see study-5	see study-5
7	*Developing Software with Scrum in a small Cross-Organizational Project:*	Difficult to estimate tasks due to high degree of R&D	---	No daily stand-up, Scrum meeting every third day	---	Sprint reviews, No mention of retrospectives
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	Challenging to balance workload during sprint planning	Challenging to track progress during sprint and to react accordingly	Challenging to build cross-functional team from initially highly specialized people. Daily stand-up meetings	Several findings identified	Several findings identified
9	*Understanding Decision-Making in Agile Software Development: a Case-study* *only about project South*	Several findings identified	---	Stand-up meeting for 15min with *PO* and *PM*, *PM* use 1-2min to inform team about meeting with potential customer.	Several findings identified	Daily meetings are needed to avoid decision-hijacking, *SM*, *PO* and *PM* separate meetings preparing backlog for next sprint.
10	*Understanding Shared Leadership in Agile Development: A Case Study:*	product backlog too detailed	Several findings identified	Daily stand-up, often dragged out, people were unprepared	Several findings identified	Several findings identified

Table 4.6: Analysis of the selected studies using the "Big Five" framework.

In the following sections, each component of the framework will be described in regards to how it is addressed in Scrum, and the findings from the selected studies related to the relevant Scrum aspects will be presented. In some cases it was problematic to place the Scrum practices reported in regards with the team leadership framework, as several practices had implications for more than one behavioral maker. However, in my opinion, the reported Scrum aspects have been placed according to the particular behavioral maker where the link was determined to be the strongest. When compiling the list of findings specific to each of the behavioral makers defined by (Salas et al., 2005), only the studies marked as "Several findings identified" in table 4.6, for the particular category, are included. In addition, it should be noted that three of the studies (5, 6 and 9 - Project North) reported from the same project, thus providing much more information regarding this particular project.

4.2 Facilitate Team Problem Solving

Facilitate team problem solving is the first behavioral maker for team leadership, as defined by (Salas et al., 2005). When considering its description in section 2.5 in regards to Scrum, it appears that it involves, among other things, to what degree the *scrum master* facilitates team problem solving by removing impediments and how sprint planning is performed. When considering the selected studies through the findings outlined in table 4.7, I found that very little of the literature was devoted to describing how the *scrum master* removed impediments for the team. However, it was reported that an open office space often (study-4 and -5) facilitated more discussions among the team members, thus leading to increased team problem solving. Although, it is pointed out (in study-4) that being situated like this made it hard to focus. Based on the findings from the selected studies, this section has been split into team problem solving during sprint planning in 4.2.1 and distributed team problem solving in 4.2.2, while a summary of the findings is given in 4.2.3.

No	Studies:	Team problem solving
2	*Customising agile methods to software practices at Intel Shannon*	Tasks split across sprints, 2 planning stages - at start of project and when starting each sprint
3	*Using Scrum in a Globally Distributed Project: A Case Study*	Sprint planning: distributed meeting, local meeting in Norway and Malaysia.
4	*The Impact of Agile Practices on Communication in Software Development*	Open office space providing efficient discussion forum but hard to focus, developers used daily meetings for design problem solving (opposed to purpose of defining project status)
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 and study-9 (Project North)*	discussions between *SM*, *PO* and developer going to do work, problem solving often not done in the team, team barely participated in decision making during planning meetings, not willing to share knowledge, the team missing the big picture due to division of work according to specialization and non-collective decision making, the team was situated in open office space
7	*Developing Software with Scrum in a small Cross-Organizational Project*	Difficult to estimate tasks due to high degree of R&D.
9	*Understanding Decision-Making in Agile Software Development: a Case-study (Project South)*	Discussions between *PO* and the team when estimating tasks and deciding what to do during sprint planning, the person who knows most decides. Issues identified early. Some specialization within team. If not solved within the team *SM* talk to *PO* and *PM*.
10	*Understanding Shared Leadership in Agile Development: A Case Study*	Sprint planning meetings: *PO* making product backlog too detailed.

Table 4.7: Team problem solving.

4.2.1 Team problem solving during sprint planning

From the selected studies it is apparent that several different practices exist when it comes to sprint planning such as; allowing tasks to be split across sprints and having two planning stages – one at project start and one each time a new sprint is initiated (study-2) as well as arranging geographically distributed meetings (study-3). Estimating tasks and deciding what to do was also carried out as a discussion between the *product owner* and the team, whereas the frequent dialog resulted in an early identification of problems, as seen in study-9. However, in some cases, the sprint planning meetings can turn into a discussion between the *scrum master* and the *product owner,* as reported in study-5. Consequently, in this case, the team barely participated in decision making during the planning meetings. When the team was involved, the person who knew the most about the topic decided what should be done. Moreover, this lead to a division of work according to specialization, thus making team problem solving very difficult. As a further consequence, the team became unaware of what the others were doing and thereby lost sight of the big picture. The result being that the daily stand-up meetings became uninteresting.

Other issues impeding team problem solving may also occur, as reported in study-10, where the *product owner* made the product backlog too detailed and thereby reduced team problem solving by constraints imposed by the backlog items. It is also problematic if the daily stand-up meeting is used for team problem solving regarding design decisions, as reported in study-4, in contrast to its purpose of providing information about project status. A good practice, however, is to try to help individuals solve their problems by discussing the issues within the team and if a solution is not found, the *scrum master* may consult the *product owner* and the project manager, as seen in study-9.

4.2.2 Distributed team problem solving

The only fully distributed project is found in study-3. In this project it is attempted to facilitate team problem solving, despite having team members geographically spread, by conducting a distributed sprint planning meeting followed by a local meeting in both Malaysia and Norway. This bipartite approach considers the fact that the teams are located remotely, but still need to coordinate their activities and attain a shared mental model as well as establish collective thinking and motivation for the team members that are

co-located. Consequently, this illustrates that team problem solving in a distributed environment is possible.

4.2.3 Summary

From this analysis it appears that an important part of facilitating problem solving is to encourage self-organization within the team. Furthermore, it is essential to establish trust throughout the organization in order not to overrun the team, but allow them to take responsibility for the planning of the tasks at hand. Self-organization may directly influence effectiveness as it brings team problem solving and decision-making authority to the level of operational problems, thus increasing the speed and accuracy of problem solving (Moe & Aurum, 2008).

4.3 Provide Performance Expectations and Acceptable Interaction Patterns

Provide performance expectations and acceptable interaction patterns is the second behavioral maker for team leadership, defined in (Salas et al., 2005). The findings from the selected studies that relate to this particular behavioral maker are outlined in table 4.8. From the table it appears that the most commonly reported interaction pattern was pair programming (study-2,-4 and -10), which is not a part of Scrum although an often used supplement known from XP. Pair programming is an interesting form of interaction as it allows for knowledge sharing thus supporting backup behavior[18] and making the team as a whole more adaptable over time. Furthermore, the team members working together can inspect each other's progress and provide aid when needed. Additionally, pair programming facilitates mutual performance monitoring[19] as a common understanding in the team environment, which contributes in making the development of suitable performance metrics less complicated. These are reasons to why subsection 4.3.1 is devoted to pair programming. The rest of this section is split into reported problems with not having acceptable interaction patterns, in 4.3.2, and how performance expectations and project tracking were performed, in 4.3.3.

[18] See section 2.5 Team Leadership for an explanation of backup behavior.
[19] See section 2.5 Team Leadership for an explanation of mutual performance monitoring.

No	Studies:	Performance expectations & interaction patterns
2	*Customising agile methods to software practices at Intel Shannon*	Pair programming widely used - lead to smaller code base, testing, refactoring, simple design, high-level tasks split to distr. across sprints, effort estimates, contingency built into plan, contingency factor tuned as project progresses, team wrote wrap-up reports giving measurements of actual sprint effort in regards to initial estimates
4	*The Impact of Agile Practices on Communication in Software Development*	Pair programming, discussions caused by the open office space, design discussions during daily meetings, close interaction between researcher team and company representatives, top managers in reviews
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 and study-9 (Project North)*	Did not want to edit each other's code, few problems discussed among developers, customer support interrupting developers work, lead to developers making individual plans, result *SM* directing -> less self-organized, little discussion due to specialized areas of work (not cross-functional), problems regarded as personal - not reported, high individual autonomy less team autonomy
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	Challenging to track progress during sprint and to react accordingly.
10	*Understanding Shared Leadership in Agile Development: A Case Study*	The team considered pair programming on complex functionality during planning meeting in sprint 5. The wall was used to make every task visible and categorized into "not started", "in progress" and "finished".

Table 4.8: Performance expectations & interaction patterns.

4.3.1 Pair programming as an acceptable interaction pattern

By employing interaction patterns such as pair programming, a smaller code base may be obtained because instances that probably never will occur are, more often, not taken into account when working in pairs (study-2). It can be argued that this gain is achieved through knowledge sharing between the two developers. A smaller code base also leads to more efficient resources usage, through i.e. less debugging at later stages, since the defect rate is directly correlated with the code length. Pair programming may be particularly suitable for difficult tasks, as indicated by study-10, where the team (after sprint 5) utilized this technique mainly for complex functionality.

4.3.2 Problems with not having acceptable interaction patterns

Teams that enforce division of work according to specialization will reduce the team's ability to self-organize, as indicated in study-5. The result of specialization is less

discussion among the team members which leads to segregation. A further consequence is developers making individual plans, which indicates lack of team orientation. As a result, the *scrum master* might feel forced to take up a more directing role, which in turn reduces self-organization within the team. The outcome is high individual autonomy[20], but a less autonomous team. Consequently, this reduces the common ownership of the product and team members will refuse to edit each other's code. Moreover, this constrains the possibilities for pair programming and other kinds of interaction patterns within the team.

4.3.3 Performance expectations and project tracking

Most of the selected studies reported that after receiving a prioritized backlog from the *product owner*, some sort of estimation process, whereas expectations for when tasks would be completed based on performance monitoring, was performed during the sprint planning meeting. Although, very few described the estimation process in detail or addressed how this process changed according to the actual results as the project progressed. Furthermore, remarkable little attention was devoted to describing well known estimation techniques such as planning poker. Study-2 reports that high-level tasks were split in order to distribute them across sprints until the duration of each sprint corresponded to at most 20 working days. According to (Fitzgerald et al., 2006); "Contingency was built into the plan and effort estimates were done based on ideal engineering effort", although how this estimation was actually performed is not addressed. Although, it is reported that the contingency factor is tuned as the project progressed.

Tracking project progress can be seen as a useful basis for developing appropriate performance expectations and is often regarded as a difficult process. When considering the selected studies, only three of them (study-2, -8 and -10) where found to report on this aspect, although it was mentioned in study-4 that top management attended review meetings. This practice may imply that the organization as a whole would be better informed about project progress and thus better positioned to perform performance expectations for the remaining parts of the project. On the other hand, study-8 describes it as challenging to track progress during a sprint and to react accordingly. This indicates the difficulties of

[20] Autonomy describes to what degree an individual is independent and self-governing.

staying agile and especially the challenges of improving the situation within a sprint where one might have underestimated the needed effort and thus find oneself behind schedule.

It should be noted, however, that two very interesting practices for tracking project progress were reported. The first one is from study-2 and involved having the team lead write wrap-up reports listing the tasks completed including extra tasks that were not part of the original sprint plan. These reports also contained "lessons learned", but most importantly measurements of the actual effort used in the sprint in regard to the initial estimates. By employing such a practice it may be a lot easier to track project progress, thus improving estimates and making performance expectations for the next sprint as a data foundation is already established. In the second practice, the task board was used to help the team track project progress and allow for mutual performance monitoring during the sprint, by making every task visible to the team as they were categorized into "not started", "in progress" and "finished". This practice, which was found in study-10, made it possible to easily see what tasks remained in the sprint, so that resources could be directed where they were needed the most in order for the team to adapt accordingly. By employing such a technique, it helps the *scrum master* become more vigilant as it increases the awareness regarding project progress. Additionally, monitoring where impediments might occur in the near future is made easier by inspecting tasks that have been "in progress" for a long time or by examining the other remaining tasks.

None of the selected studies addressed focus factor or team velocity which are metrics for performance expectations. This might be an indication of frequent changes of team members, since a stable team is needed to develop these metrics. Another reason may be that the majority of the teams were not mature in the use of Scrum, given that the focus in the first phases, when running a Scrum project with a fairly new team, revolves around more fundamental aspects of Scrum than performance expectations. It also takes some experience from the *scrum master* to see the importance and have the competence to calculate appropriate metrics. Additionally, a fairly well established process with stable sprint-length and the needed data are required.

4.4 Synchronize and Combine Team Member Contributions

Synchronize and combine team member contributions is the third behavioral maker for team leadership as defined by (Salas et al., 2005) and in regards to the Scrum framework,

the daily stand-up and the review meeting are the main practices relevant to this behavioral maker. The reason being that the review meeting is an evaluation of the team members' synchronized effort throughout the sprint, whereas a sprint delivery is demonstrated by combining the team member contributions. However, I have decided that the findings related to review meetings should be addressed in section 4.6 as they are more relevant to this category which delves into practices concerning meetings and feedback sessions. Consequently, the findings from the selected studies, which are outlined in table 4.9, revolve mainly around the daily stand-up meeting. As a result, the rest of the section is split into continuous integration, in 4.4.1, the role of the daily stand-up meeting, in 4.4.2, and problems encountered during the daily stand-up meeting, in 4.4.3.

No	Studies:	Synchronize team member contributions
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction*	Daily stand-up meetings (only they were seated), kept customers up to date on the progress of the software and informed on issues as they happen rather than at the end, helped customers stay in the loop and have a better idea when to expect questions
2	*Customising agile methods to software practices at Intel Shannon*	Daily stand-up meeting around task board with post-it notes, moved notes around and into 'done' area, prepared notes before meeting
4	*The Impact of Agile Practices on Communication in Software Development*	Continuous integration gave info on status of end product.
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 and study-9 (Project North)*	Daily stand-up increased team communication, *SM* overreacting to problems, *SM* acting more like a *PM* than a coach, not listening or talking to each other during daily-stand up meeting, too much focus on what have been done and too little on what people are going to do
9	*Understanding Decision-Making in Agile Software Development: a Case-study (Project South)*	Daily stand-up meeting for 15min with *PO* and *PM*, *PM* use 1-2min to inform team about meeting with potential customer.

Table 4.9: Synchronize team member contributions.

4.4.1 Continuous integration

A helpful scheme may be to employ continuous integration throughout the project, because this practice allows the team to synchronize and combine team member contributions so that the current state of the product can be inspected, as seen in study-4. When being able to deploy the latest code to a test environment on a regular basis, customer feedback can be attained in a timelier manner, which greatly increases the chances of meeting the customer needs and wants, thus achieving customer satisfaction. Additionally,

by constantly monitoring how a project progress, increased awareness can be realized in regards to potentially needed changes, which may lead to more agile team dynamics.

4.4.2 The role of the daily stand-up meeting

In the daily stand-up meeting, the team members present what each of them have contributed with since the last meeting, what their impediments are, if any, and what they will do towards the next meeting in order to keep the team up-to-date and synchronize team effort. Details concerning the daily stand-up meeting are reported from projects discussed in study-1, -2, -5 and -9. The daily stand-up meeting can be seen as a way to keep customers up-to-date on the progress of the software and informed on issues as they happen, rather than at the end – as reported in study-1. The meeting can also help customers to stay in the loop and have a better idea when to expect questions. In order to make it easier for the team to synchronize and establish an overview, by combining individual team member contributions, the daily stand-up meeting may be held around a task board covered with appurtenant post-it notes describing what is to be completed in the current sprint. Notes can be moved into a "done" area as they are being completed. This practice was employed in study-2, although in order to facilitate this meeting as effectively as possible, each team member had to prepare their own notes (often as post-it notes), before the meeting. By utilize this practice the meetings also became more interesting for the team members, since they all came prepared.

4.4.3 Problems during the daily stand-up meeting

When people work on tasks according to specialization, they may stop listening or talking to each other during the daily stand-up meetings as the problems and the status of the other team members no longer seems to concern them. This happened in study-5 whereas it complicated the process of synchronizing team member contributions. However, the daily stand-up meeting increased team communication, when compared to the situation before Scrum, but it did not work as intended. One of the reasons being that the team members were also afraid of taking on more responsibility as customer support kept interrupting their work. Consequently, the *scrum master* started acting more directing towards the team members, thus taking up a role with more resemblance to a project manager than a coach. This had a negative effect as it further reduced the team's ability to self-organize, thus

making synchronization of team member contributions solely the responsibility of the *scrum master*.

Another mistake regarding the daily stand-up meeting is that too much focus can be directed towards the work performed since the last meeting, as reported in study-5. Developers may want to keep this part of the meeting short, while placing more emphasis on what should be done towards the next meeting, as this has more practical use to them. Consequently, it should be avoided to give the meeting a more retrospective angle than desired at the expense of synchronizing and combining individual team member contributions towards the next meeting and the completion of the sprint backlog.

Improved communication flow within the organization can be obtained by having both the *product owner* and the project manager attend the meeting, as reported in study-9. This practice is rather unusual as only the team and the *scrum master* normally attend the meeting, and in some cases the *product owner*. During the daily stand-up in study-9, the project manager always used 1-2 minutes to inform the team about meetings, if any, with potential customers. As a consequence of this practice, the project status is at any given time distributed to high-level management through the project manager. However, this extra intervention, in the daily stand-up meeting, also has a downside as new requirements and features demanded by the market may lead to expansion or changes to the sprint scope, as seen in study-9. In most cases this means that the sprint should be cancelled, although information about such action is not found within the case study. The fact that more stakeholders were kept up to date during the daily stand-up meeting should suggest that synchronizing and combining individual team member contributions would be easier, but since this often lead to new requirements entering the sprint this practice must be considered as a two-edged sword at best.

4.5 Seek and Evaluate Information that Affects Team Functioning

Seek and evaluate information that affects team functioning is the fourth behavioral maker for team leadership defined by (Salas et al., 2005). When considering how this behavioral maker is described in section 2.5, it appears that the burndown chart is one of the most important information resources for the team. The reason being, that it provides the team members with enriched information in order for them to develop similar mental models and make informed decisions which will align them towards completing the sprint goal.

Even though the burndown chart can be considered one of the prime information sources affecting team functioning during a sprint, as seen from table 4.10, only two of the selected studies (study-5 and -8) reported using this artifact. Based on the fact that the burndown chart is such an important artifact to the team, it may be experienced as problematic when the burndown chart is not updated regularly. This problem was reported in study-5, and clearly illustrates the importance of maintaining the information affecting team functioning. Further on, the rest of this section is split into findings related to the wall/task board (in 4.5.1), the product vision (in 4.5.2), the product backlog (in 4.5.3), user stories (in 4.5.4) and alternative communication channels given in 4.5.5.

No	Studies:	Info. affecting team functioning
2	*Customising agile methods to software practices at Intel Shannon*	Contingency plans, tasks published online, post-it notes on a board
4	*The Impact of Agile Practices on Communication in Software Development*	Task board, Proj 1: mailing-list to communicate with the whole team Proj 2: user stories, short daily status reports on e-mail, info sharing between R&D and management ← project status.
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 and study-9 (Project North)*	Missing proper vision & proper communication with *PO*, false deadlines, burndown chart not updated regularly,
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	Challenging for *SM* to balance between enforcing and caring roles. Difficult for *PO* to prioritize and present product vision. Used a burndown chart. Physical size of task card indicating work load, card flipped upside down to indicate changes
9	*Understanding Decision-Making in Agile Software Development: a Case-study (Project South)*	Product backlog seen as a way to make *PO* prioritize tasks, the backlog makes the team aware of the long- and short-term goals of the project – avoid the hold-up problem.
10	*Understanding Shared Leadership in Agile Development: A Case Study*	The wall; making every task visible and categorized into: ”not started”, “in progress” and “finished”. No mention of burndown charts.

Table 4.10: Info. affecting team functioning.

4.5.1 The wall/task board

The wall/task board, which normally also contains the burndown chart, is maybe the most useful information source for the team. It is reported to be actively used in four of the selected studies (study-2, -4, -8 and -10). Post-it notes describing the different sprint tasks may be placed on the wall and some tasks may even be published online, as carried out in study-2. An interesting practice that could be adopted by other companies in the industry is

to use task cards of various physical sizes in order to indicate the work load of each task instead of estimating tasks in hours, as reported in study-8 and seen in figure 4.1. This makes it possible to get an overview of the required effort through a quick glance. In particular, task cards may have three different sizes, as seen in study-8, whereas the largest indicated a work load of 4 (a big task), a half-card was a normal task indicated by the number 2 and a quarter-card was a small task given by the number 1. The numbers may then be used when calculating burndown charts. In addition, task cards may be flipped upside down when something is changed in order to make it easier for the *scrum master* to follow up the changes, as practiced in study-8 and seen in figure 4.1. When it comes to structuring the task board, every task can be made visible to the team and additional overview is offered by categorizing each task into e.g. “not started”, “in progress” and “finished”, as reported in study-10. This practice allows for improved monitoring of the internal team environment and facilitate effective information search and evaluation, thus helping the team track project progress during the sprint. Consequently, the team adaptability can be improved by allowing suitable team behaviors to be carried out.

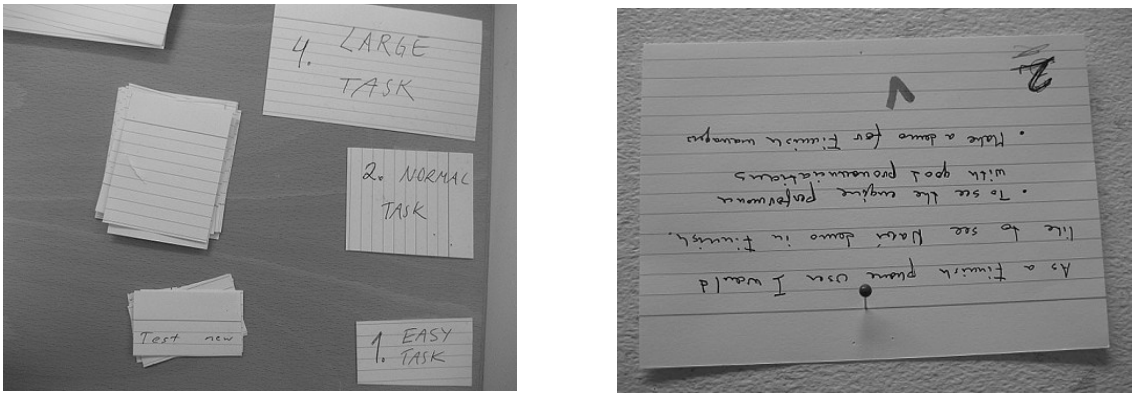

Figure 4.1: Task cards in different sizes and a task card flipped upside down.

Source: (Marchenko & Abrahamsson, 2008)

4.5.2 Product vision

The advantages of establishing a product vision as an informative guidance to the teams is an aspect of Scrum and software development in general, that has not been devoted a lot of attention in the selected studies. This is indicated by two studies (study-5 and -8) as they reported that communication and presentation of a proper product vision was not in place.

4.5.3 The product backlog

In many cases, the product backlog might serve as one of the most valuable sources of information affecting team functioning. It may also have an additional function towards ensuring that the *product owner* would have to prioritize the tasks to be performed, as reported in study-9. Apart from that, the product backlog informs the team of the long- and short term goals of the project, instead of this information being hidden in the architecture, which is commonly referred to as the hold-up problem (Freeland, 2000) in the literature and often seen in traditional projects. By keeping the product backlog readily accessible, increased transparency throughout the organization can be attained, thus making information, which may affect team functioning, more available.

4.5.4 User stories

According to Mike Cohn (Cohn, 2004), user stories is an effective and appropriate technique for processing information found in the product backlog items as a team and to determine how each item should be broken down into tasks before being distributed during sprint planning meetings. As a result of having worked more thoroughly on the backlog items when developing the user stories, the team members are better informed with regards to how the current sprint backlog will affect team functioning. Among the selected studies, only one (study-4) specifically reported on the utilization of user stories (in project 2). This does not necessarily mean that this was the only project using this technique, but may rather imply that the researchers behind the studies did not choose to emphasize this aspect when writing the report. On the other hand, user stories might indicate a more structured Scrum approach which might only be achieved after having gained some experience with the methodology. Therefore, the reason to the under-documented use of user stories might be due to the low maturity levels of the Scrum teams.

4.5.5 Alternative communication channels

Communicating and transferring information that may affect team functioning can be orchestrated in a range of different ways such as through the task board or meetings. However, yet another communication channel within the team may be to utilize mailing-lists to continuously distribute information to the whole team instead of using individual e-mails, as reported from project 1 in study-4. Short reports on the project's daily status may also be shared with external stakeholders through e-mail, a practice adopted in project 2 from study-4. By sharing information in this push-based manner, people that take interest in the project is kept updated on the overall status in a more timely fashion. In particular for project 2, in study-4, it facilitated information sharing between R&D and the management, while ensuring that the management was kept up-to-date on the project status.

4.6 Engage in Preparatory Meetings and Feedback Sessions with the Team

Engage in preparatory meetings and feedback sessions with the team is the final behavioral maker for team leadership as defined by (Salas et al., 2005). As seen from table 4.11, almost all of the selected studies reported on preparatory meetings and feedback sessions. The most common form of preparatory meetings is the sprint planning meeting. However, the daily stand-up meeting can be seen as both a preparatory meeting and a feedback session. The reason being, that each team member gives the rest of the team and the *scrum master* feedback on what was done since the last meeting and what impediments, if any, they are experiencing in regards to their work - thus being a feedback session. Additionally, plans for the work towards the next meeting are shared with the rest of the team and the *scrum master* – thus making it a preparatory meeting. That being said, the most common feedback sessions are the review, the sprint retrospective and the Scrum of Scrums meetings. However, who attends these meetings and how the meetings are run seem to differ across the selected studies. The outline for the rest of this section is new preparatory meeting practices in 4.6.1, while daily stand-up and sprint planning preparatory meetings is given in 4.6.2. Further on, review and retrospective feedback sessions are found in 4.6.3 and the Scrum of Scrums feedback session in 4.6.4, while a summary of the meetings in general is presented in 4.6.5.

No	Studies:	Meetings and feedback sessions
1	*A Case Study on the impact of Scrum on Overtime and Customer Satisfaction*	Customers involved in planning, daily stand-up & reviews, review offers ability to tweak & change product. developers may demonstrate their effort in review & retrospective, developers receive questions/suggestions for improvement, planning meeting reduced customer confusion about future tasks, *SM*/*PM* & customer not well prepared -> meetings drag on, solution *SM*/*PM* started planning between sprints to help customer formulate ideas to help developer estimation –> prepared & effective planning. Daily stand-up meetings kept customers up to date on progress & informed on issues, in the loop & when to expect questions, daily stand-up & retrospective meetings could drag on if not focused enough
2	*Customising agile methods to software practices at Intel Shannon*	Wrap-up report of tasks including "lessons learned", demos, releases and project reviews. planning at the start of project and when starting each sprint
3	*Using Scrum in a Globally Distributed Project: A Case Study*	Daily stand-up 15 min with *PO* and one meeting room on both sites with teleconference connection and Web cameras, sprint planning meeting divided in 3 parts: distributed meeting during 3 synchronous working hours with *PO* explaining backlog items followed by consecutive site-specific parts, dist. phase arranged using teleconferencing and application sharing, Sprint review – team, *SM* and *PO* demos through teleconferencing & application sharing, Retrospective – 1h, using teleconferencing & application sharing weekly Scrum of Scrums – 0.5h, teleconference & web camera
4	*The Impact of Agile Practices on Communication in Software Development*	All stakeholders involved in sprint planning meetings, top managers in reviews, retrospectives, documentation workshops with stakeholders
5	*Scrum and Team Effectiveness: Theory and Practice* *reported info also taken from: study-6 and study-9 (Project North)*	Did not use the feedback mechanisms of retrospective properly, did not act upon changes, *PO* did not attend all retrospectives, *PO* joined some of the stand-up meetings by telephone, planning meetings often ended in discussion between *PO* and *SM*, people showing up late for meetings not 100% committed focusing on their own modules, not listening or talking to each other during daily-stand up meeting, too much focus on what have been done and too little on what people are going to do
8	*Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*	A team decided to exclude *SM* and *PO* from retrospectives, more comfortable, less issues resolved, in one team most demos had to be accepted before the review
9	*Understanding Decision-Making in Agile Software Development: a Case-study (Project South)*	Daily meetings are needed to avoid decision-hijacking, *SM*, *PO* and *PM* separate meetings preparing backlog for next sprint.
10	*Understanding Shared Leadership in Agile Development: A Case Study*	Low use of retrospectives in studied company. *SM* was too instructive during meetings.

Table 4.11: Meetings and feedback sessions.

4.6.1 Preparatory meetings: new practices

It should be noted that some studies (study-1 and -9) report on interesting preparatory meeting practices performed in the name of Scrum, while not part of the original methodology. Study-9 describes a separate meeting initiative between the *scrum master*, *product owner* and the project manager to prepare the product backlog for the next sprint. A similar practice is reported in study-1 after the developers remarked that the *scrum master*/project manager and the customer were not prepared enough for the planning meetings so the meetings dragged on. Consequently (in sprint 8 – the last sprint), the *scrum master*/project manager took on a business analyst role between sprints by actively starting the planning and long term visions ahead of time together with the customer to help them formulate their ideas with enough detail in order for the developers to use them when estimating. By employing such a practice, it may ensure that the *scrum master* and the customers are better prepared for the planning meetings, thus leading to an easier and more effective process.

Often the sprint preparation job is performed by the *product owner*, however, by involving higher-level staff as the project manager (as seen in study-1 and -9), this might provide a wider timeframe to the prioritizing. Furthermore, by including the *scrum master* a closer link to the team and knowledge of what they are capable of in terms of work load and technical difficulty are attained, thus increasing the probability of a more suitable prioritizing of the product backlog.

4.6.2 Preparatory meetings: daily stand-up and sprint planning meetings

By letting customers attend the daily stand-up meetings, they are kept up to date with regard to project progression and informed on issues as they happen rather than at the end, as seen in study-1. These meetings also help customers stay in the loop and have a better idea when to expect questions. However, the meetings can drag on if they are not focused enough, as reported in study-1.

Customers may find that attending planning meetings reduce confusion about what is being developed, while developers often find the sprint planning meetings useful since they can choose the scope of work. Although, guessing the timeframe for each backlog item is not easy and time estimates do not always come out right, as pointed out in study-1. In order to

better facilitate project planning, an additional preparatory meeting can be introduced. Consequently, each project may have two preparatory planning sessions; one in the beginning of each project and one at the start of each sprint, thus introducing a new kind of meeting when initiating a project. This was carried out in study-2, whereas each team lead made a plan outlining all the sprints to the end of the project during the preparatory meeting at the start of the project. By employing this practice, the initial meetings can be conducted to attain high-level estimates for allocation and distribution across several sprints. Furthermore, dependencies between the tasks within a sprint could be worked out in advance, but might prove to be difficult at this stage as well as time consuming. In addition, trying to plan far ahead also contradict the Scrum principles, that is, focusing on planning for only one sprint at a time. It should be noted, however, that working out dependencies during the first preparatory planning session were not practiced in study-2.

4.6.3 Feedback sessions: review and retrospective meetings

Furthermore, as it is difficult to visualize the product ahead of time and some concerns only arise during the demo, review meetings offer customers the opportunity to see the product at an early stage giving them the ability to tweak and change the product in a timely fashion. Furthermore, review meetings allow developers to demonstrate accomplishments and to prove their own accountability in regards to the tasks that they undertook. Additionally, developers may also receive useful questions/suggestions for improvement during these meetings. Furthermore, by having top managers attend the review meeting (as reported in study-4) this may suggest easier tracking of project progress, which in turn may lead to increased ability to stay agile as information are flowing more freely through the organization, thus making it easier to act upon needed changes.

An alternative to the customary retrospective meeting were reported in study-2, whereas the team lead wrote a wrap-up report, when finishing a sprint, listing the tasks completed including extra tasks that were not part of the original sprint plan. In addition, the report also contained “lessons learned” and a measurement of the actual effort used in the sprint in regards to the initial estimates. Although the report-writing replaced the customary retrospective meeting, the demos, releases and project reviews were customary at the end of a sprint. Further on, retrospectives may often not be focused enough and and thus tend

to go on for too long, as reported in study-1, which indicate that a report-writing practice may be a valid alternative to the customary retrospective meetings.

4.6.4 Feedback sessions: Scrums of Scrums

Study-3 was the only one that reported on the use of Scrum of Scrum meetings. It was conducted as a half-hour distributed meeting using teleconference and web cameras with one representative from each team and all *scrum masters* attending. As the project in this study was distributed, the demos in the sprint review meetings were in some cases done through teleconferencing and application sharing. A similar practice was reported in study-5, in which the *product owner* was situated in another city. In that particular project the *product owner* joined some of the stand-up meetings by telephone.

4.6.5 Summary

The problem with most meetings seems to be that the participants often struggled to stay focused, and as a result the meetings dragged on. Based on the selected studies it seems that in order to keep the meeting short and focused the participants need to be as prepared for the meeting as possible.

5 Empirical Multi Case Study Results

The first part of this chapter provides context to the companies in which the interviewees are employed as well as background information to each project. The rest of the chapter presents the results identified when examining the interviews by utilizing the team leadership framework, given in section 2.5. Moreover, all the findings obtained are categorized according to the five behavioral makers for team leadership, which is defined as one of the "Big Five" in (Salas et al., 2005), whereas each category is represented by one section, 5.2 – 5.6. However, it should be noted that practices relevant to the behavioral makers of team leadership were not reported from every project, thus only findings relevant to the framework are presented in this chapter.

5.1 Research Context

The table below characterizes the projects investigated, through the seven interviews that were conducted, according to project size, degree of distribution and customer interaction in order to determine how the extent of the project affects project management. By using the same dimensions as in section 4.1.1, to characterize the projects, it will be possible to compare how the results from the empirical case studies compares to the findings identified in the literature.

No	Project:	Project size:	Distributed project:	Customer interaction:
11	*Project-11:*	Project length: 2.5 years Teams: 7 Team size: 4-9 Sprint length: 2 weeks	Distributed between Norway and Malaysia	Internal *PO* *PO* involved in pre-planning, attended review meetings and was a part of the reference group. All customers could report identified bugs in Jira (issue-handling system)
12	*Project-12:*	Project length: 2 months Teams: 10 Team size: 7-9 Sprint length: 3 weeks	Not distributed	Internal *PO* – a group of analysts from the organization takes on this responsibility leads to problems with ownership and authority for running the operation. Two full-time resources from the customer are also involved. Spend typically 4hours with the customer during sprint planning.
13	*Project-13:*	Project length: 1.5 years (+0.5 years before Scrum) Teams: 1 Team size: 7-9 Sprint length: 6 weeks	Not distributed	No *PO* product backlog more or less defined by the existing system. Users inquired developers regarding desired functionality

14	*Project-14:*	Project length: 1.5 years Teams: 3 Team size: 9 Sprint length: 4 weeks	Not distributed	Internal *PO*
15	*Project-15:*	Project length: 3 years Teams: 8-9 Team size: 6-7 Sprint length: 4 weeks	Not distributed	External group (design team) taking on the *PO*-role
16	*Project-16:*	Project length: 5 months Teams: 1 Team size: 7 Sprint length: 4 weeks	Not distributed	Internal *PO* *PO* attended most of the daily stand-up meetings. Short project, but the team member consultants were mature in the use of Scrum.

Table 5.1: Categorization of empirical multi case study results.

The interviewees report from projects performed by in total four different international IT consulting companies, whereas three of them have well over 15 000 employees in more than 15 countries. The final and smallest company (in size) has a bit more than 1 000 employees and is established in 6 countries. The companies specialize in consultancy services, technology, integrated services, development and maintenance and outsourcing. All the projects reported were carried out in Norway.

Project-11 separates itself from the other projects because a product company is in charge of the project, which means that the company develops a certain (software) product and each new major release (version) is considered to be a new project. Each project lasts for about half a year (as there is approximately one year between each release) and the company itself have been using Scrum for about 2.5 years. Since each project involves development of the same product, a rather stable working environment for the teams is established. Furthermore, the teams in this project are regarded as the most mature, of the projects examined in the empirical multi case study, as they consist of several individuals that have been working according to Scrum for more than 2 years. This may create fertile conditions for adaptation and evolvement of Scrum practices. During this period of time the sprint length has been changed from four to two weeks. The product is developed for the oil industry and is meant for tracking and classifying oil specifications. At the start of the project, the teams were relatively stable, but after a little while replacements started to occur. However, the teams seems to have stabilized around 7±2 members. Furthermore, the organization's objective for the future is to have stable teams over time, preferably over

several months. On the overall aspect of how well the organization was adapting in regards to Scrum, one of the *scrum masters* remarked that:

> "I feel that we have not reached the full potential of Scrum, yet. However, we are continually doing improvements, adapting and slowly getting better for each sprint."

Project-12 involves developing a big case handling system for the public sector and has been running for two months, but despite being short lived, this project is considered highly mature as several people involved in the teams have experience with Scrum from other projects. However, the second company, which is in charge of this project, has officially been using Scrum for one year. The core of the Scrum methodology, as viewed by the company, is divided into three groups by 3/4/3 elements. This is best illustrated as follows:

Roles:
3→ *product owner*, *scrum master*, the team.

Ceremonies:
4→ Sprint Planning, Daily Stand-up Meeting, Review, Sprint Retrospective.

Artifacts:
3→ Product Backlog, Sprint Backlog, Burndown Chart.

Every Scrum project in the organization involves these elements, thus if a project contains these elements it is considered to be a Scrum project. In addition, it is required that the project is agile, implicating that the practices employed in the project are adapted according to the customer's needs, requirements and organization as well as to the company's organization. However, the last element is must harder to gauge or pinpoint. There might be some deviations from this model as the customer often defines the Scrum concept for the organization, thus how Scrum is run may differ slightly across projects. A normal team in the organization typically consists of 8-9 people including the *scrum master* and most teams have been stable for over a year.

The third company was in charge of three projects (project-13, project-14 and project-15) and has officially been using Scrum for about 1.5 years. Based on experiences from different projects the organization always use fixed sprint length and if several teams exists, they always work in parallel using the same sprint cycle. The reason being, that this

introduces a certain rhythm to the project, thus ensuring that everyone involved knows when sprints are completed, when to synchronize and so on.

Project-13 was an embedded project that revolved around developing a road informatics system. An old system existed, but a new one was required. The project has been completed and lasted for 1.5 years, which means that the team involved can be considered mature. 6-week sprints were employed as the sprints were lead by each function being developed and nothing less than six weeks could be used in order to complete all the tasks related to the given functionality. The team consisted of 7-9 developers (team size kept close to constant), where most of them were specialists within their own domain. Consequently, no more than two people could overlap each other's work, thus reducing self-organization by restricting the freedom of choice when delegating the work during the sprint planning meeting. This was something they wanted to change by trying to obtain more people within the same knowledge area. The *scrum master* acted as a project manager for everything external to the project such as management, customers and users. On the other hand, internally to the developers he undertook the role of a *scrum master*.

Project-14 lasted for 1.5 years and ended in January 2009, implying that the teams involved can be considered mature. The *scrum master* introduced Scrum to his team, the test-team, already from the beginning of the project. In total, three teams were involved in the project whereas the other two were running RUP methodology. The test-team consisted of 9 members at most, although the team started as 3-4 members in a phase where no more people were needed and was expanded throughout the project and then scaled down a bit. The changes in team size were done according to work load. However, only two people from the company were involved in the project at all times. 4-week sprints were used as this was perceived to be the de-facto standard and it seemed to fit in with the adjacent projects and variations of this were only occurred in connection to vacations.

Project-15 has been running for about three years now and is considered to involve highly mature teams. The *scrum master* who reported from this project has only been involved in the project the last five months, however, he has previous experience as a *scrum master* for 1.5 years from project-14. The project revolves around developing a large new pension system for the public sector. Further on, the sprint length has been kept constant at four weeks during the time the *scrum master* has been working on the project.

Project-16 started in August 2007 and revolved around helping a customer develop a software product for the shipping industry. Scrum was introduced to the customer through this project and most team members were relatively new to Scrum when the project started. Morover, the developer reporting from this project left the project after five months, thus the team members did not get the time to become mature during this period. However, the developer interviewed has worked on several Scrum projects since project-16 and have reflected on a lot of the practices used this project, thus making his input valuable nonetheless. Throughout the project a sprint length of four weeks were kept constant and the team involved seven developers, one *scrum master* and one internal *product owner*.

5.2 Facilitate Team Problem Solving

The facilitation of team problem solving was identified to be strongly connected with two aspects of Scrum, namely; sprint planning, given in section 5.2.1, and the handling of interruptions and external requests, which can be found in section 5.2.2. The reason being, that one of the main arenas for team problem solving is the sprint planning meeting, whereas the team breaks down the user stories found in the product backlog, perform estimation, discuss how to solve tasks in the upcoming sprint and possibly consult the *scrum master*, the *product owner* or other resources. Further on, handling external inquiries appropriately is crucial as a reduced team complicates the establishment of shared mental[21] models, coordination[22] and lowers the team's collective motivation[23], which decomposes the environment for team problem solving.

5.2.1 Sprint planning

One adaption reported from the industry after having run Scrum for a while, is that teams may refrain from breaking off a sprint if for instance only minor changes in scope are made, as reported from project-11. If following Scrum by-the-book, such a change would impose discontinuation of the current sprint. It is also quite common in the industry, that user story acceptance criteria are not properly defined before initiating the next sprint,

[21] See section 2.5 Team Leadership for an explanation of shared mental model.
[22] See section 2.5 Team Leadership for an explanation of coordination.
[23] See section 2.5 Team Leadership for an explanation of collective motivation.

which has been a problem for a long time in project-11. However, they are working on this problem and one of the developers remarked that:

> "A critical area for improvement was developing satisfactory requirements before starting a new sprint. However, this demands time dedicated to this purpose by an already overloaded *product owner*."

A solution is to focus more on developing pre-defined and very specific acceptance criteria for each task during the sprint planning, before initiating a sprint, which is currently an area in focus in project-11. The acceptance criteria in this project are written on the form: User can [select/function] so that [output] is [visible | complete etc.] and an example of an acceptance criterion is: "User can read document with pros/cons." Furthermore, the acceptance criteria may be established jointly through discussions between the team and the *product owner* to alleviate the last-mentioned part as well as to enhance the quality of these criteria, as newly practiced in project-11. The main advantage of having more specific acceptance criteria, is that it is easier for the team members to know when a certain task can be considered complete. The criteria also helps the *product owner* when deciding whether to accept a task or not after a demo, during the review meeting. For each sprint in project-11, the user stories, with appurtenant acceptance criteria, were placed on the task board below the task-matrix as seen in figure 5.1.

Figure 5.1: The maintenance team in project-11 is located in the same room.

An adaptation of Scrum can be to make the *product owner*-role obsolete, as seen in project-13 (however this is rarely done and not necessarily recommended). In this project, most of the system was already defined through the existing system, thus it was not considered difficult to determine what to make. Additionally, the architect of the old system was a part of the team and as he knew the old system, he had a good idea of what to prioritize at any given time. During the sprint planning, the team examined the product backlog and as they were always working in arrears, the items concerning new functionality were prioritized based on what the customers kept demanding.

Similar to project-13, where most of the system where already defined, the product backlog was also rather fixed in project-14 as a typical sprint for the test-team involved preparing tests for the deliveries received from the development department. Moreover, since the team was working on testing, they were dependent on input from the development, service and maintenance teams. Consequently, review and planning meetings were conducted on the same day when these resources were available. Estimation was performed as a team, although planning poker was never used, while three-point-estimation (estimating a task with a min, max and most likely value) was performed to some degree at first. When the team became more experienced, focus was directed towards what was done in previous sprints during the estimation process and adjustments were made based on knowledge and beliefs. In comparison, the team in project-13 went through the items that were added to the sprint backlog and estimated in detail the tasks for each person in the team during the planning meeting. These estimates were then inserted into a spreadsheet in which also the burndown chart was maintained. The same spreadsheet was always used and a sprint number was added to each task, which enabled filtering the tasks due in the current sprint. Consequently, the same spreadsheet was used for product backlog, sprint backlog and the burndown chart. In project-15, on the other hand, a thorough estimation process was already performed before the project started, when trying to win the project. As a result, the product backlog already contained estimates, a practice that deviates from the Scrum principles. However, the team made their own estimates when planning, but the pre-made estimates acted as guidelines none the less. In contrast, the estimation process in project-16 did not involve the same degree of team problem solving. Instead, the team members who understood the product the best decomposed the tasks and performed the estimation. Consequently, the team as a whole did not attain ownership of the product. Another result was that those who possessed the most competence and was the

most enthusiastic chose the preferred tasks, while the more "boring" tasks were left to the passive team members. Furthermore, during the estimation process, planning poker was used at first, but its use was discontinued because the team members who failed to understand the product contributed with estimates that were way off, according to the consultants. Even though, one of the main reasons to running planning poker, according to (Kniberg, 2007), is to clear up any misunderstandings in regards with the work that is to be undertaken during the sprint, one of the developers remarked:

> "…there was no point (in running planning poker) as the whole team did not comprehend the problems."

However, when a problem had to be solved during a sprint, in project-16, it was discussed internally within the team. The ones responsible for the task sat down and thought through the problem and solved it together. If they still had not solved the problem and they realized that they could benefit from input provided by the other team members, it was brought up during the daily stand-up and a meeting with the right people was arranged. If the problem persisted, a spike lasting a couple of days after the sprint was arranged where the team members could read through relevant literature or turn to other sources. It was common practice to have a couple of days between each sprint for this sort of issues. When problems related to the domain occurred, the *product owner* got involved and as he had office on the same premises and worked full time on the project, he was not very busy and thus available to assist the team with problem solving at any time.

In contrast, to using available time to indulge in relevant literature, as in project-16, team members with free capacity, in project-13, were working on activities related to the next sprint, from time to time. This practice violates typical Scrum-principles, and was one of the Scrum-modifications made during the project. In particular, team members could be working on a report that was needed later, although this work was not due in the current sprint. This kind of work was not included in the sprint backlog as it would not be a part of the current sprint delivery. The reason why they did not rather focus on helping out team members in the current sprint was probably due to the high degree of specialized work.

The projects varied greatly in terms of agility, but even though most of the system in project-13 was already defined by the existing one, some agile adaptation were made, through team problem solving as sudden requirements were met, while down-prioritizing

other functionality in the backlog. As an example, when pictures taken of number plates from cars were not read automatically, it required a certain application to be installed on the particular computer which had to be run over a secure line towards a server located in another city. This was regarded as a bad solution and despite being behind schedule and in the need of more team members, the team solved this problem by making a web application with the same functionality that could run from any location. In comparison, the team in project-14 was extremely agile, according to the *scrum master*. The reason being, that the test-team had a very strict deadline for when the testing of the different parts of the system should be completed. This often involved acceptance testing with the customer as well. Both the customer and the organization (the team was a part of), was waiting for the testing to be completed in order to move on to other activities. As a result, there was almost no other alternative, than to deliver on time. Problems were also solved relatively quickly in project-15, much due to the daily Scrum of Scrums meeting (more on this practice in section 5.6.4), which made sure that issues were handled at the appropriate level as soon as they were identified. The frequency of this meeting contributed to increasing the agility of the project as a whole. In contrast, the team in project-16 was not agile enough due to either the solution outlined by the management not being good enough or because the team did not fully understand how the desired solution should be developed, according to one of the developers. The developer further explained that the team was very vulnerable to changes since more practice with test-driven development was needed;

> ”In order for the team and the software to be agile you have to run test-driven development. Only then is it possible to maintain code that may change without causing any particular problems.”

Different software tools may be used in order maintain the product backlog and facilitate the sprint planning, as seen in project-11, whereas Jira was used to administrate the back-log, issues and to report bugs, while the sprint planning was facilitated by a web-based tool called Target Process (which was installed on a web server locally) in project-16. Target Process also had different purposes and was used by the *product owner* to add user stories to the product backlog and as a task board. The sprint planning itself took at most three days much due to the management’s high focus on the products flexibility. Consequently, the discussions during the sprint planning were often very technical, as a result many of the

team members from the customer side were not able to follow or take part. Therefore, the high-level format of the discussion prevented real team problem solving.

5.2.2 Handling interruptions and external requests

Customer support caused team members in project-11 to be pulled out of the team and the result was smaller and less effective teams. Additionally, whenever a person is removed, so is the competence of that individual. However, the advantage of having two people from different teams swap places is that new impulses can be attained and experiences from the newcomer can be shared across the team. This allows for some degree of knowledge sharing between the teams. Further on, the organization has increased the focus on cross-functionality. From now on, a team will work on different product areas with assistance from several product managers (the organization defines a product manager as a person responsible for one of the product areas. There may be several product managers serving a team, but only one *product owner*). As an example, the maintenance and production team has been merged, consequently providing the team with two product managers, but only one of them act as a *product owner*. The team-merging was performed in order to facilitate knowledge sharing and making the project less vulnerable to sickness, team changes or people quitting the team. This configuration also provides wider background knowledge among the team members, thus facilitating better conditions for team problem solving.

The problem with external inquiries were also familiar in project-13, project-14 and project-16, whereas the teams were constantly bombarded by customer inquiries, meetings, bug fixing, and other issues that had to be prioritized. The team in project-13 also had to do sales support in connection to selling the solution to new projects. As a result of all the external requests, some sprint goals were not reached, which reduced the team's collective motivation[24]. As team members where regularly pulled out of the project it was difficult to establish shared mental models[25] and coordination[26] within the team, which reduced team problem solving within the team. The *scrum master* in project-14 remarked that the most important task in regards with his role was to protect the team from external requests. An

[24] See section 2.5 Team Leadership for an explanation of collective motivation.
[25] See section 2.5 Team Leadership for an explanation of shared mental models.
[26] See section 2.5 Team Leadership for an explanation of coordination.

attempted solution in all three projects was to direct all external inquires through the *scrum master* in order for him to deal with them first, which solved some problems, although this did not always work. According to the *scrum master* in project-13 this may have been caused by the sprint length in this project:

> "A disadvantage with having such long sprints (6 weeks), is that external inquiries cannot wait until after the sprint. They had to be brought in and handled mid-sprint in many cases. If we were running shorter sprints, we could have responded that the resources would be available after 3-4 days, when the sprint was over, so they just had to wait until then."

The scrum master in project-16 said something similar:

> "One of the disadvantages of running such long iterations (4 weeks), might have been that team members were taken out of the team every month to work on other tasks not related to the project."

In an attempt to amend the constant inquiries for issues such as support and bug fixing, a service window defining the amount of time to be used either per day or per week was established in project-13. According to the *scrum master*, this was the best solution they came up with in order to prevent the external inquiries from pervading the team members' daily work. In project-14 the *scrum master* also took the issue concerning external requests to a *scrum master*-level in an attempt to establish a routine where resources where scheduled to be borrowed for x hours at the time. Through this approach, it was possible to adjust the burndown chart to reflect hours the team members were not working on the project, which allowed an overview of hours not available to the project. Another solution to the problem could be to let the *scrum master* accumulate all these small inquiries to fill one or two days, and then finish them all off by releasing the resources needed, as seen in project-16. This was not practiced at first, but the team (in project-16) realized fairly quickly that certain team members were too frequently interrupted. It also became common practice, among the people working other projects, to request resources from the *scrum master* through e-mail. These changes made the team more stable and created a better environment for discussion and team problem solving.

5.3 Provide Performance Expectations and Acceptable Interaction Patterns

Several performance expectations and acceptable interaction patterns were identified in the Scrum projects, whereas the main interaction patterns were XP techniques and especially pair programming, given in section 5.3.1 and Scrum training, presented in section 5.3.2. Performance expectations and tracking of project progress were found to be performed with help from burndown charts (treated in section 5.3.3) and by calculating team velocity (described in section 5.3.4).

5.3.1 XP and pair programming

Although not part of Scrum, pair programming may be used as a technique for ensuring better code quality. In the organization in charge of project-11, it was company policy that all code should be peer reviewed or produced through pair programming. Both of these interaction patterns facilitates mutual performance monitoring and backup behavior as common understandings of the problems at hand are developed and the teammates may assist each other both verbally and behaviorally. Pair programming may also be utilized to bring new team members or people not familiar with a certain technique, language or tool, up to speed, which also increases the total competence level within the team, as seen in project-11 and -16. The reason being, that pair programming facilitates knowledge sharing among the two individuals working together, which is also helpful to create a flying start when working with a big or rather complex task, as reported from both project-11 and -16. It was reported that pair programming was especially useful when introducing test-driven development to the team in project-16. In addition, according to the developer interviewed, digressions and procrastination could be avoided.

However, working in pairs may be experienced as a strenuous process as individuals most often have different work rhythm and habits as well as break-patterns. That was the reason why pair programming was only encouraged when it was assumed to contribute with increased efficiency or product quality, in project-11. Furthermore, people tend to find out who they work well with in particular settings, thus the *scrum masters* in project-11 did not encourage pair swapping all too often, since unnecessary changing of pairs would only impair effectiveness. As the teams were small, the members of each team would have, after some time, been working with all the other team members nevertheless. This would be a result of pair swapping occurring naturally, thus allowing knowledge sharing across the

team. However, in one of the teams in project-12, there was a somewhat conscious process of making the members in the team work with everyone else in the team in order to facilitate increased knowledge sharing. In that particular team, people were asked to switch pairs from time to time. However, in the other teams, in this project, *no* such practices exist. A similar mindset existed in project-16, where no guidelines were established in relation to the use of XP-techniques such as pair programming, and management was not involved in how software development was carried out. In this project, pair programming was seen as beneficial interaction pattern and one of the developers remarked that:

> "Pair programming is a nice technique for cooperation and useful for competence transfer as well as a good way to sustain focus when working on a particular task."

It is apparent that several companies in the industry considers interaction patterns such as pair programming advantageous as the organization in charge of project-12, where it was common practice for the teams to pair program for about 3 hours every day. This was not demanded, but desired from the teams and most of the teams decided that they wanted to pair program when the *scrum master* brought this up. There was also a strong focus on other XP-techniques such as test-driven development, continuous integration, small working deliveries and refactoring. Especially testing received a lot of attention within the organization and was seen as an important technique to attain a good design. According to the chief architect, extensive testing also made the team spend less time on debugging and made it easier to maintain other people's code. By having all team members construct tests for the code they were working on, it alleviated people, to some degree, from specific roles within the team as a team member could for instance be both a tester and a developer. This enforced team autonomy by contributing to making the team a role in itself, which is in accordance with Scrum theory.

Another way of encouraging discussions and knowledge sharing may be to situate the team members so that two and two developers, working on somewhat similar tasks, are facing each other, as seen in project-13. In the long run, this practice might contribute to backup behavior within the team. However, pair programming was not used, although there was a very direct dialogue between the pairs facing each other. If, say, one of the two people working on the user interface were inexperienced and needed assistance, the conditions facilitated face-to-face communication, thus the location of the developers was very good in terms of internal communication and encouraged conversations and internal meetings.

Pair programming or other techniques known from XP was not used in project-14 either. However, this was mainly because the team was working on testing and not development. Although, it would have been possible to use pair programming, but even though the quality of the product may have been improved, it was perceived that the low number of resources (this was a small project) and the high time pressure did not allow for the (possible) extra time spent on pair programming.

5.3.2 Scrum training

When the organization in charge of project-11 started using Scrum the *scrum masters* were trained and certified, in addition all the employees were initially sent to courses. However, according to one of the *scrum masters,* the focus on proper training was not sufficient in the early phase. All the employees hired after the initial stage had to rely on training from their *scrum master* and by their fellow team members. Furthermore, the *product owners* were forgotten and thus left untrained. The organization simply underestimated the importance of the *product owner*-role. The result was some misunderstandings and less effective utilization of Scrum. After about 2.5 years, they came to realize the importance of having *product owners* with a thorough understanding of the Scrum process and internal courses were held whereas the *product owners* were certified. During the last few years, the organization has also hired a consultancy company to inspect how they do Scrum, identify problems and potential for improvement in order to facilitate process enhancements. This intervention has made them more aware of the different aspects and elements of Scrum, and kept them on track. In addition, it has served as an extra form of training for new employees.

As the organization had an office in Kuala Lumpur (KL) in Malaysia, working on the same projects as in Norway, they truly practiced distributed Scrum. This further complicated the coordination of the Scrum activities. The chosen solution has been to send representatives to the KL office from time to time in order to facilitate face-to-face meetings. This was seen as a way of creating a closer bond between the two offices, improving communication, exchange practices and perform some training. However, these encounters took place rather seldom, and some time passed before they realized that some of the problems, experienced when working with the KL office, were due to lack of understanding regarding the Scrum process itself. The main reason was probably insufficient training of

the employees and the fact that the *scrum masters* for the two distributed teams was located in Norway. In order to remedy the situation, the employees at the KL office were given more training, but most importantly several people became certified as *scrum masters* during this process. The outcome was an increased understanding of Scrum and as a consequence the communication between the two countries was enhanced. Through this experience it is evident that appropriate and acceptable interaction patterns must be in place, in a distributed project environment, in order to maintain a shared understanding of the Scrum process and to synchronize and combine team member contributions.

In contrast, there was a very low degree of official Scrum training within the project-12. They relied mainly on knowledge transfer through teamwork and working in pairs as the way of spreading both Scrum practices and information about technology within the teams. The emphasis on Scrum training was also fairly low in project-13, whereas the training process consisted of describing a short version of what they were doing in the project and then presenting an introduction of Scrum called Scrum Awareness. After this initial introduction, the team just had to pay attention and learn-by-doing. The Scrum Awareness introduction was also used in project-14 and project-15, as it was carried out by the same organization. Furthermore, the *product owner*, in the project-14, was internal and did not go through any formal training, but seemed to adapt as the project progressed, according to the *scrum master*. Most of the people working on project-15, however, were employed in the organization and thus already familiar with Scrum through the introduction course and other projects. However, some of people on the project received *scrum master* certifications externally during the project. The *product owner*-role has handled by an external design team. Whenever a team finished working on a module, it was approved by the individual, within this design team, who was responsible for this particular module. In such cases, this individual took on the *product owner*-role. However, it is unclear whether the individuals in the design team had received any form for training prior to undertaking this role. However, it should be noted that dividing the *product owner*-responsibilities in this manner is not in accordance with the Scrum principles.

In project-16, the *scrum master* came from the customer side and was trained by a Scrum certified consultant and as he was also a team member the *scrum master* was continuously trained throughout the project. The rest of the team members from the customer side received an introduction course, lasting a couple of days, about how to run Scrum when the

project was initiated. After having run Scrum for two sprints, two project managers from the consultant side who had been using Scrum on another project with great success held a presentation. During this presentation they explained what had been working for them and what did not. According to the developer, this interaction was very instructive and informative, and increased the team's understanding of the Scrum process, which boosted performance through competence and knowledge transfer.

5.3.3 Performance expectations: burndown chart

An easy way to calculate and maintain the burndown chart is to simply draw it on the task board next to the tasks and update it during the daily stand-up meetings, as seen in project-11. This also made the burndown chart visible and readily available to the team members as they were all located in the same room, and allowed them to clearly see how the sprint progressed, which provided a good indication of the teams expected performance for the rest of the sprint. In project-13, however, the daily stand-up meetings were primarily used for reporting status and not for activities related to the burndown chart, when first starting to use Scrum. Similarly, a burndown chart was not used during the two first sprints in project-14. When the chart was introduced, the team calculated the burndown in a shared excel spreadsheet where all the team members filled in the hours spent on each module in the project and estimated the time remaining for the tasks they were working on. This was common practice in project-13,-14 and -15, while an additional top-level excel spreadsheet was made for each team in project-15 to monitor the overall progress for the modules they were working on. However, this practice imposes additional work for the team members in terms of having to keep the spreadsheet up-to-date with remaining hours. Some teams did not use a task board, instead the spreadsheet was used as the common medium and a print out of the burndown chart was placed on the wall during the daily stand-up meeting, as seen in project-13 and -15. However, when looking back the *scrum master* (in project-13) admitted that it would have been advantageous visually to have had a task board where one at least could move the main tasks around.

The daily stand-up meeting in project-16 was also always completed by showing the current burndown chart. This practice made the team velocity apparent to the team members on a daily basis. This could be motivating, but also alarming as the team might be informed that several hours and much effort remained until the team was back on track

according to the sprint delivery. Furthermore, both the *scrum master* and the *product owner* were present during the daily stand-up meetings, which made identifying and initiating needed measures a quick process.

In order to alleviate some of the additional work imposed by keeping the spreadsheet up-to-date in regards with time spent on developing tasks for each module, the team in project-16 had a database from which hours reported to be used on the project were automatically exported into the system for payments. This practice allowed them to have only one place to record working hours thereby avoiding problems related to double booking, which saved the team valuable time. Consequently, it made the recording process less cumbersome, which encouraged the employees to update frequently, thus keeping the burndown chart closer to the actual values. The consultants, however, had to record the hours used in two systems, but the employees in the company benefited greatly from this practice. Furthermore, the system was extensively documented considering that a fine-granularity down to task-level was employed.

5.3.4 Performance expectations: team velocity

Story points produced during a sprint may be used as a metric for team velocity, as seen in project-11. During the project the teams were rather unstable, but lately they seem to have stabilized, thus making it possible to estimate team velocity. One of the *scrum masters* reported that the team he was in charge of was currently able to produce 0.75 story points per day, per person. The term story points was introduced after the team made the transition from estimating tasks, during the sprint planning, in days to story points. In contrast, hours were used when estimating in project-13, thus hours was the metric used in the team follow-up as well. The team follow-up constituted a measure that indicated team velocity. As an example, the tasks for a given sprint might have been estimated to take 500 hours to complete. After the sprint, it may have been identified that 1000 hours were used instead. This gave an indication of how to adjust the estimates for the next sprint and thus a suggestion of team velocity. The number of tasks completed was also examined, thus revealing another indication of the amount of work the team was able to produce during a sprint.

At first the team (in project-13) often used twice the amount of estimated time. In such cases, they expanded the sprint with one week (or more), making it an irregular sprint lasting for seven weeks. This practice violates Scrum principles, but was considered

necessary as the project depended on the team delivering working functionality. The sprint was only terminated when the planned functionality was completed. Consequently, it was not possible to maintain a constant sprint length of six weeks. Reasons to these expansions can be traced back to the fact that only one or two team members were often able to perform a certain task and with constant external inquiries, it was difficult to complete the goals within the time frame of an ordinary sprint.

According to the *scrum master*: "This project was not ideal for Scrum, but having delivery iterations, daily meetings with status updates, sprint planning and retrospective meeting are so positive by its own means that it is beneficial to include them in the project. It allows one to think what went well and what did not. "

5.4 Synchronize and Combine Individual Team Member Contributions

Synchronization and combination of individual team member contributions were mainly performed through the daily stand-up meeting, seen in section 5.4.1, and continuous integration schemes, described in section 5.4.2. However, synchronization and combination of individual team member contributions is also a prerequisites when preparing the product before the having the review meeting, but as practices employed during the review meeting will be treated in section 5.6, it will not be considered in this section.

5.4.1 Daily stand-up meeting

Synchronizing and attaining an overview of team member contributions can easily be facilitated during the daily stand-up meeting by moving around tasks on a task board to illustrate progress in the project, as seen in project-11 and illustrated by figure 5.2 (the depicted task board was not used in project-11). This also enables mutual performance monitoring and coordination across the team. A similar practice was also reported in project-16, however, a projector showing the different tasks were used and tasks were moved around using a computer. The result being that the overview of the individual team member contributions was not visible once the meeting was over and the projector was turned off, instead the team members had to access a website. Compared to the approach from project-11, this practice reduced the availability and tangibility of the information.

Figure 5.2: Tasks being moved on a task board in mid-sprint.

Normally only the team and the *scrum master* attended the daily stand-up meeting in project-11, while in project-13 a technical project manager and an individual responsible for infrastructure were also attending. Even though a lot of people took part, they still managed to keep the meetings within 15 minutes. The individual responsible for infrastructure was present as he could often help to answer questions that arose during the meeting. According to the *scrum master* in project-13, the daily stand-up meeting was considered to be the most effective aspect of the Scrum methodology.

A spreadsheet was available to all the team members (project-13), and one of its functions was to reflect the remaining work for the different tasks in progress. This was especially important for maintaining an up-to-date burndown chart of the sprint (which was also maintained in this spreadsheet). However, after a while it became apparent that people were not very good at keeping the spreadsheet updated every day in regards with work left to complete. At first, the team decided to update once a week, but later agreed to do this every day. In practice, however, most people ended up updating remaining work on their tasks every second or every third day, instead. This was seen as a major challenge when trying to synchronize and combine team member contributions.

Even though focus was on having each team member update the remaining work, people did not have to explain in detail during the meeting why they, for instance, did not make any progress since the last meeting. If having to report in such detail, the team members

might have been pushed more towards performing, however, lengthy explanations would have made the meeting exceed its time frame. Additionally, lack of progress was usually caused by impediments imposed by external inquiries, which were commonly known across the team.

5.4.2 Continuous integration

Although not part of Scrum, a continuous integration process could combine individual team member contributions by retrieving the newest source code, compile it and run unit tests, as initially performed in project-16. However, this process may be expanded by allowing the compiled code to be deployed to a test environment which, everyone in the project and the customer may access, as seen in project-11,-12 and -16. This makes it possible to inspect the current state of the product on a daily or even hourly basis. The consequence of this practice is that both the team members and the customers are provided with enriched information facilitating the development of shared mental models[27]. This is important to align the team and helps to meet the customer needs, as feedback from the customer may be received in a timelier manner. However, there was mixed feelings and skepticism, among the team members on the customer side in project-16, regarding the return on investment of running continuous integration at first. The reason being, that they did not see utility value of this initiative. However, after a while especially the *product owner* learned to appreciate the value of continuous integration as he could get an overview of the current state of the software and the developers did not have to do deployment manually which ensured a more streamlined process. This vindicated the extra time and effort spent on maintaining this process to both the *product owner* and the rest of the team. Furthermore, it was reported that both continuous integration and test-driven development were employed in the project-15. In contrast, no such practices were employed in project-13 as the project consisted of porting an old system to a new platform and these kinds of challenges. However, according to the *scrum master*, such techniques would have been a part of the process if they were allowed to start from scratch, but employing these practices in an old system was considered very challenging. In project-15, however, these practices were initiated at an early stage allowing a build server to be used

and whenever someone checked in code that could not build a red light was turned on automatically and a picture of the individual that checked the code was displayed on a big screen. This was deemed necessary as several teams often depended on one individual's code. As a consequence, this initiative made sure that people spent more time looking for errors in the code before checking it in, thus increasing code quality.

Automated testing, which is not mentioned as a part of Scrum either, is often considered a vital part of any continuous integration scheme, and reported to be used to a large extent in project-12 and in the organization in charge. A lot of time is saved by not having to carry out tests manually, according to the chief architect. It also gives the team the opportunity to check if everything that used to work before the new build still have the correct functionality, after the new changes and updates have been added to the codebase. This, in turn, allows them to construct nightly builds which contributes to a more seamless continuous integration.

A useful practice for synchronizing the individual team member contributions and improve awareness of team contributions between teams may be to perform peer reviews each time a module is due to delivery, as was common practice seen in project-15. In this project, a part of the definition of done for a module was to have a co-worker from another team walk through and approve a checklist that outlined everything from code standards to use of framework. A peer review was arranged by scheduling a suitable time with someone from another team. Everyone, but the novice developers (people who had just joined the team) could perform a peer review. In addition to the peer review, it was also required to have a certain amount of documentation, such as Javadoc, in the code.

5.5 Seek and Evaluate Information that Affects Team Functioning

In regards with seeking and evaluating information that could affect team functioning, the Scrum projects revealed that the teams had many sources to turn to and the main ones given in this section are; the wall/task board (in 5.5.1), product vision and press release from the future (in 5.5.2), the product backlog (in 5.5.3), the sprint length (5.5.4) and other sources (5.5.5).

---

[27] See section 2.5 Team Leadership for an explanation of shared mental models

5.5.1 The wall/task board

A good practice to quickly provide an overview on the status of the sprint tasks, is to write them down on post-it notes and place them in a matrix according to degree of completion and move them around as progress is made, as seen in project-11 and -15. This allows the team members to easily seek and evaluate information about project progress. As a result this may affect team functioning by for instance directing more resources at a difficult task that have been in progress for a long time in order to for it to be completed. A similar practice was employed in project-16, whereas a digital task board was used and displayed to the team, during the stand-up meeting, using a projector. However, after the daily stand-up meeting, the projector was turned off and then a website had to be used to view or update task status through re-estimation or reporting remaining hours for each tasks. This made the information less available to the team, thus increasing the threshold for seeking and evaluating sprint progress.

In project-11, the task-matrix had the columns "next", "in progress", "to be approved", "done" and the amount of story points involved, as seen in figure 5.3. Similarly, in project-16, the first column showed all the tasks not started, and the others showed tasks "in progress", "ready for test" and "complete". Additionally, in project-11, the tasks in progress had a picture of the team member(s) working on it, as seen in figure 5.3. Furthermore, a drawing of a house depicted the customer and pictures of the team members working on-site for the customer, were placed on this house. The use of the pictures quickly allowed anyone to determine what each team member was doing at any given time. The user stories relevant to this sprint were also placed on the task board, below the task-matrix, which allowed the team members to be reminded of for instance the acceptance criteria or how to demo any particular story. All these practices were not initiated at the start, but throughout project-11 increased emphasis was being placed on using the task board. According to one of the developers:

> *"We are now using the wall to a much larger extent, which gives the team a better overview of the process. In particular, the fact that the burndown chart is visible on the wall at any point in time makes the progress more visual to the whole team and this makes the developers more eager to finish their tasks."*

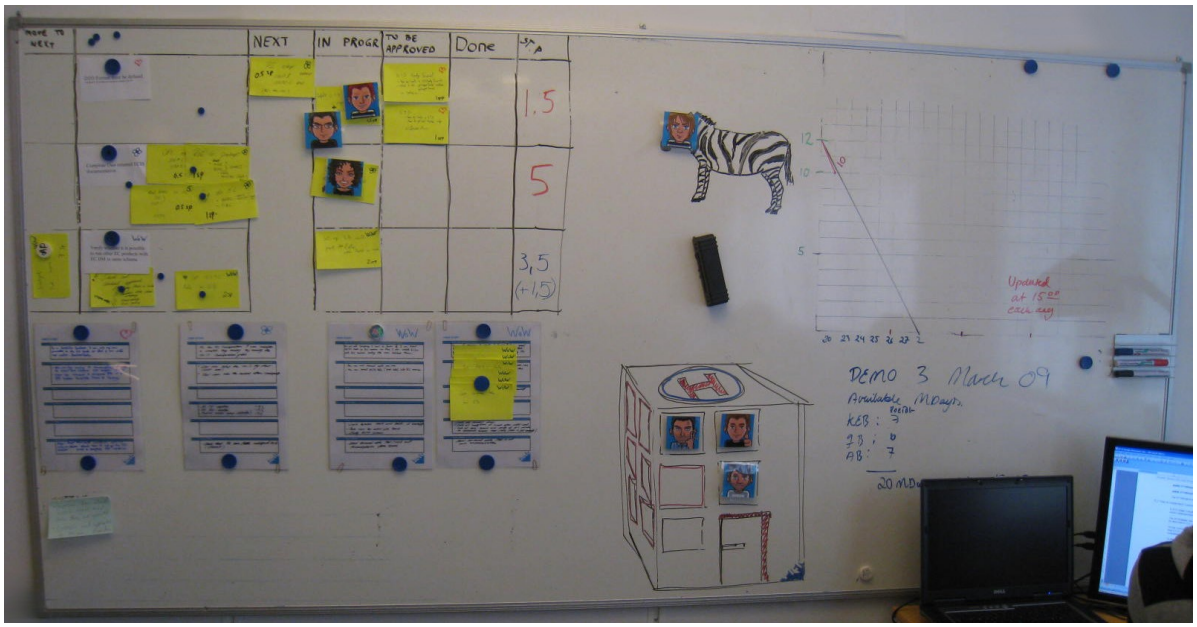

Figure 5.3: The task board and burndown chart in project-11.

An additional useful practice could be to have a separate section of the task board for issues, as seen in project-15, whereas red post-it notes were used for this purpose. It was also common practice within project-15 to have each team member prepared red post-it notes with issues and the number of hours spent trying to solve this issue, if any had occurred, in order for them to be put up and explained during the daily stand-up meeting. The hours spent trying to solve the issue was important as it gave the *scrum master* an indication of how much time was wasted on non-production related work.

Similarly, in project-13 a list of issues was also placed on the wall along with a list of tasks that were due and the individual responsible for each, but post-it notes were not used. In contrast to the other projects, the wall itself was only maintained by the *scrum master*. A print out of the burndown chart was also placed on the wall along with a simple project plan attached to a brown paper. The project plan indicated important dates to the test-team as for instance future deliveries and main activities such as when acceptance tests were due. This was an initiative from the *scrum master* in order to create a better understanding and to present the team with the big picture.

5.5.2 Product vision and press release from the future

Establishing a distinct product vision for each project that defines the goals to be achieved and indicating the project stakeholders may serve as an informative guidance for team

functioning, as seen in project-12. In the organization in charge of project-12, it was customary, that every project had a precisely defined vision for the delivery. An appropriate and much used way of developing a projects vision was through a technique called "press release from the future", according to the chief architect. This technique was used to understand how a successfull project would look like. It consisted of four parts:

- Course partner – declared how much that was spent.
- Managing Director – declared how important it was to spread the knowledge possessed within the project.
- Fictional course participant – how important it was for him/her.
- Fictional manager – how important it was for him/her.

This technique allowed the teams to be more conscious about the means and instruments needed to make the project a success.

5.5.3 The product backlog

In project-13, all the Scrum elements were not used. In particular, there was no *product owner* (see section 5.2). The reason being, that the customers knew less about the system than the developers. Consequently, the developers had to prioritize the product backlog. However, this was not a seen as a problem as the system requirements were very detailed and a logical order of deliveries existed, thus making the *product owner*-role rather redundant. The size of the product backlog ensured that the developers would not run out of things to do in the foreseeable future and the developers kept prioritizing the work to be included in each sprint according to what the users needed the most at that time.

5.5.4 Sprint length

When the team has become comfortable with the running Scrum it may be beneficial to decrease the sprint length, as seen in project-11 where this was performed after running Scrum for more than a year. For a long time the sprint length was set to one month. However, during these sprints a lot of support suddenly emerged all too often. This made valuable members of the team unavailable for long periods of time. In addition, the demand for quick bug fixes, major patches or service pack releases required valuable

resources from different teams. In order to properly coordinate these releases, the sprint length had to be reduced. By reducing the sprint length to fourteen days, which is what is currently used, it made it easier to postpone interruptions until after a sprint was finished and to adapt the next sprints with upcoming service releases. This works well according to everyone who has been interviewed. A shorter sprint length also allowed for problems to be identified at an earlier stage, in order for the necessary measures to be taken. The result was more agile project management. Additionally, one of the developers remarked:

> *"...having shorter sprints with smaller goals, which in turn is easier to achieve is more motivating for the team. The shorter sprints also make it easier to adapt to unforeseen changes."*

Other important consequences of having shorter sprints include reliable velocity-metrics being developed more easily and the product backlog will be maintained more often, which increases its quality and improves the planning horizon. Additionally, issues that arose during the sprint are easier to remember when attending the sprint retrospective. This is obviously vital as it is difficult to fix an issue that has not been brought up. Since the sprints are very short, it is also important to keep the planning short to avoid overhead. Both *scrum masters* (in project-11) said it was company policy to try to limit the planning to one hour per week for each sprint. As a result two hours will be the maximum time spent on planning for a two-week sprint. Nevertheless, one of the developers said that the sprint planning could often lead to overhead. Furthermore, one of the *scrum masters* admitted that, in most cases, approximately half a day had to be used for sprint planning, although this was something that they were trying to improve.

In some cases, on the other hand, the sprint length is determined by the organization, as seen in project-12 whereas the sprint length was set to three weeks. The reason for having 3-week sprints was that it allowed for a demo environment that closely resembled a production ready state. Consequently, only a few teams changed the sprint length more than once a year. It was also common practice that a sprint did not end on a Friday and commence on the upcoming Monday. The breaks between sprints were short and the following sprint usually commenced the day after the review and sprint retrospective meetings.

5.5.5 Other sources of information affecting team functioning

Smart solutions and practices to common problems can be distributed across the team by utilizing wikis, blogs or discussion forums in order to effectively seek, evaluate and reuse information that could affect team functioning on a regular basis, as seen in project-11 and -16. This also enables the team members to write about problems they experience and meetings they have had as well as post figures, pictures of the task board etc. At first, a wiki was used within most teams in project-11, although after a while the use of blogs was introduced instead. The reason for using blogs was that they were considered to represent a low-threshold service, which all team members would feel comfortable contributing to. According to one of the *scrum masters*, a blog contrasts itself from wikis and similar services where people tend to be strict, careful and formal. Based on prior experience, one of the *scrum masters* claimed that team members often refrained from sharing information due to these perceived quality norms attended with posting information on wikis. Blogs also allows the team members to sort information according to keywords, the date and so on, which also aid the team members when trying to seek and evaluate information that may prove helpful for the tasks at hand.

An alternative is to use a discussion forum, as seen in project-16. However, the developer reporting from the project remarked that:

> "...I wish that this was a wiki since I have worked on projects in posterity where wikis have been used with great success. We now call this a developer handbook. It usually contains the expected code-style, how to retrieve things from the source code controller, how to commit your code, information about test-driven development, what tools we use, links to have how to use the tools and so on."

To complement the use of the discussion forum, in project-16, a whiteboard (placed on a wall) indicating the remaining working days left of the sprint and the number of days left to the first release was also used. This practice had a very positive effect, according to the developer, as especially the decreasing number of days left of the sprint helped motivate the team members. The whiteboard was updated by the *product owner* on a daily basis. Further on, the developer remarked that he would have liked to have a burndown chart that was continually updated and placed in the office landscape in order to give an even more

detailed overview of the sprint progress as this is considered to be important information that would affect team functioning.

In order to facilitate seeking and evaluation of Scrum related topics on a regular basis, to keep the team members up-to-date, new books may be purchased from time to time and spread across the team rooms, as seen in project-11. A similar practice existed in project-16, whereas books were recommended for the developers on the customer side by the consultants, which created discussions and increased awareness of the techniques to be used in the project. The books helped the team members, in both projects, to align and develop similar mental models for further education and provided guidance when in doubt regarding certain aspects of the methods used or best practices concerning e.g. user stories.

5.6 Engage in Preparatory Meetings and Feedback Sessions with the Team

Preparatory meeting and feedback sessions with the team were practiced in all the projects. The most common forms of preparatory meetings were pre-planning initiatives, described in section 5.6.1, and sprint planning. However, sprint planning practices have already been presented in section 5.2 and will not be treated in detail here. The daily stand-up meeting can be viewed as both a preparatory meeting (when each team member informs the team what he/she will do towards the next meeting) and a feedback session (when reporting what was done since the last meeting), but will not be dealt with here as it was covered in section 5.4. The feedback sessions identified that will be treated in this section, are review and retrospective meetings (in 5.6.2), the reference group (5.6.3) as well as Scrum of Scrums and other meeting practices presented in 5.6.4.

5.6.1 Preparatory meetings: pre-planning

In order to reduce the time spent on sprint planning, a pre-planning phase may be added to the Scrum framework, as seen in project-11. Pre-planning was performed during the current sprint as a way of preparing for the next sprint. It involved the *product owner* sending a prioritized list of items from the backlog to each team member, who considered the task at hand, how to break it up and made some initial estimates for each task and sent it back to the *product owner*. Based on the feedback from the pre-planning phase, the *product owner* usually made a draft which the team used as a starting point when planning the

sprint. The team members' participation prior to the sprint planning ensured that every team member was highly prepared for the meeting.

In project-12, on the other hand, the product backlog was normally prepared by the *product owner* and a business analyst. The product backlog was organized with each backlog item being a user story, but it also contained non-functional requirements involving performance such as fast services or low response times. In the past, the product backlog appeared to be unstable from time to time, due to specific maintenance routines not being defined while requirements kept being added with no specific rules being enforced to keep order. Further on, the sprint planning meeting was typically scheduled to last for four hours, but according to the chief architect, experience has shown that a whole day tend to be needed. However, it is desired to establish a parallel process for pre-planning in the future, as seen in project-11, in order to make planning more effective and save time.

During the actual sprint planning meeting user stories[28] are usually fleshed out from the sprint backlog items. The purpose of a user story is to remind the developer(s) working on it, to communicate with the customer or *product owner* in order to further clarify the particular task. The details behind the story will normally become evident during the conversation with the *product owner*. If the team members greatly disagreed when trying to estimate a particular task, planning poker can be used to resolve the problem, as reported in project-11 and -12. In both projects, planning poker was seen as a way to establish a dialogue and involve everyone in the estimation process, which in turn helped the different parts understand each other and the tasks at hand. Then the user stories were broken down into tasks. After this was done, it was common practice in project-12 to allow the *product owner* to leave, and if questions arose after he had left a choice was made based on a hypothesis. Both the choice made and the hypothesis was written down, so that the *product owner* could be consulted at a later stage, when available. An example of a user story from project-11 can be seen in figure 5.4.

[28] For more information about user stories see section 2.2.1

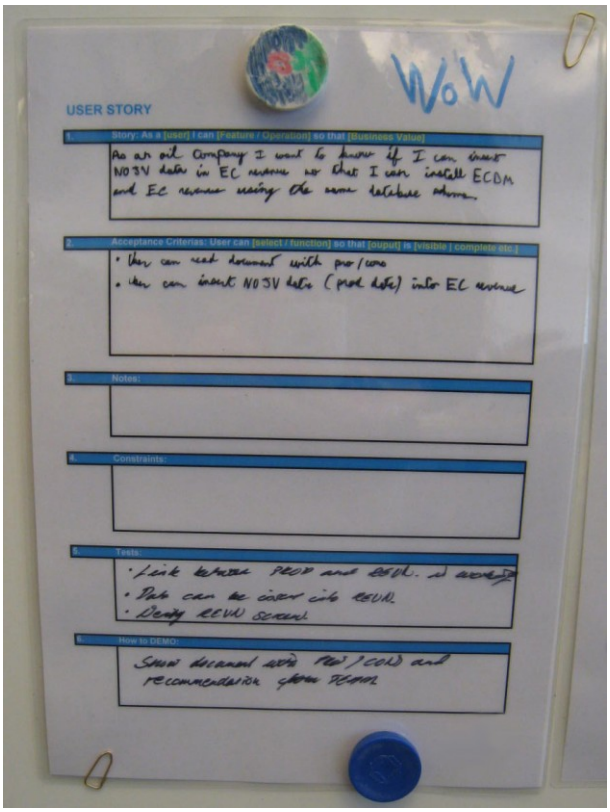

Figure 5.4: An example of a user story from the task board in project-11.

In order to fully complete additional user stories during a sprint the whole team may work on the same user story, if possible. This was practiced in project-11 and yielded a better starting point for the next sprint. Furthermore, acceptance tests are often used to confirm that a user story has been completed in a satisfactory manner, as reported in project-11 and -12. A part of the DONE-criteria, for a user story in project-12, was that the tasks involved had to be documented. However, it still remained to decide what the required documentation really was.

5.6.2 Feedback sessions: review and sprint retrospective

The review and sprint retrospectives last typically about one hour each, as reported from project-11 and -13, while in some cases longer, as in project-16 where the retrospective meeting lasted 2-3 hours or in project-14, where the review and the sprint planning were performed on the same day – using half the day for each meeting. During the retrospective meeting, it may be beneficial to let each team member tell how they experienced the last sprint while illustrating this through an energy curve on a whiteboard, as seen in project-11. The reason being, that this has a psychological effect on the human mind, causing a

person not to lie when drawing the curve (Derby & Larsen, 2006). A useful practice may be to let the team members use post-it notes for the feedback regarding what went well during the sprint and what did not, and one-by-one place them on a wall while explaining each note, as practiced in project-11 and -16. This often leads to discussion among the team members which may facilitate team orientation and shared mental models. In order to quickly be able to identify the sign of a given response and to get an overview of the sign allocation of the feedback a useful practice may be to use green post-it notes for positive feedback on the sprint and red/pink post-it notes for the negative ones, as carried out in project-11. Afterwards, the team may try to extract the essence of what worked and what did not work through a verbal summary. However, in project-16, the problem was that the meeting ended here. The team members became aware of the issues and some measures were taken in the next sprint, usually from what the team members remembered, but what was actually written on the post-it notes was not documented. There was no memo. In the end the developer remarked:

> "…maybe a good practice would have been to just leave the brown paper and the post-it notes hanging on the wall so that everyone in the team could be reminded of what worked out and what did not."

Even though the feedback from the retrospective meeting is not always documented, some changes are usually made, as in the project-13, whereas new tools to complement the shared spreadsheet were introduced. Team members were given job lists, outlining the tasks to work on hereafter, thus structuring the input concerning what to do for each team member. According to the *scrum master*, this was considered inconvenient for the project, but to the team members and the company this was advantageous as the tasks appeared more structured.

The attendees during the retrospective meeting are normally only the team and the *scrum master*, as seen in project-11,-12,-13 and -16. However, in order to better coordinate teams and facilitate knowledge sharing a useful practice may be to introduce the concept of retrospective of retrospective meetings, as recently employed in project-12. The first such meeting was scheduled on the very same day as the interview was conducted. Initially, this was carried out with two teams who work closely together. Each team first have their sprint retrospective separately and then join forces to have a retrospective meeting across the two teams.

The people attending the review meetings tend to vary, but the team, the *scrum master* and the *product owner* are normally always present. In project-12, case handlers and business executives also attended, while in project-16 the four owners in the customer's organization typically also participated. In some cases, however, no demonstration known from typical review meetings was conducted, as seen in project-13 (except for in the first sprint); the completed functionality was just delivered straight to production. This also violates the principles of a typical Scrum project. The reason for not demonstrating the product of each sprint might have been that the customer was not the user of the system, and thus not particularly interested in a demonstration. However, acceptance testing of the system was performed according to the requirements specifications after the planned functionality for a sprint had been developed. In total there were 700 requirements that were established before introducing Scrum. Each requirement was documented as it was satisfied. The acceptance tests normally approved rent, invoices, administrative aspects such as logo and customizing of the system. When the two large acceptance tests were performed, the team walked through a checklist, although there was never any form of approval from the customer.

In contrast, no acceptance tests were specified, in project-16; the team only explained what functionality they had completed and what they had not been able to finish, and then a demonstration was conducted with the customer. The continuous integration mechanisms deployed the newest version of the product, and it was shown, using a projector, to the customer by demonstrating the functionality that had been implemented. Then the progress towards implementing the flexibility desired by the customer in regards with changing processes, data structures and models in the system was discussed. A problem seemed to be that those team members who were most active during the other meetings were the ones that ended up discussing with the management. The company had four owners, which were all in the management, but all of them were not present at every review meeting. Consequently, the team did not always receive all the input needed.

Similarly, in project-15, the completed modules had to be demonstrated, although in this case, to the individual in the design team that was responsible for this particular module (taking on the role as a *product owner*) in order to get the delivery approved. It was a part of the definition of done, that each team member should demonstrate the module they had completed. In a sense this can be seen as a small review meeting in which functionality is

demonstrated. However, another form of review meeting also existed, which was merged with the retrospective meeting and thus handled as one meeting. In the first part of the meeting, which only the team and the *scrum master* attended, the team discussed benefits and concerns; what worked, what did not work and possible initiatives to make it work. At the end of each sprint a large demonstration was also arranged, which was the second part of the meeting. This meeting often involved users, project management from the customer and sometimes project management internally to the organization. During this shared official demonstration functionality delivered by all the teams, that could be presented, was demonstrated to 30-50 people. This practice enabled awareness of project progress and allowed mutual performance monitoring across teams.

5.6.3 Feedback session: reference group

In order to discuss how to handle difficult questions that the *product owner* cannot answer on his own, meetings consisting of hand-picked senior individuals and domain experts who are very often unavailable (busy) during regular work hours can be conducted, as practiced in project-11. This group of individuals represented sort of a reference group and constituted a stakeholder in its own, while representing an element not usually part of Scrum. The group arranged meetings once a week where they discussed difficult issues and made very detailed decisions regarding for instance standards for the GUI or the fraction of Java Script to be used. However, as of today, only one team receives guidance from the reference group, but this might be due to change in the future. When this team has incorporated the recent adaptations, these changes will be pursued in other teams.

A somewhat similar alternative may be to have a group of people that specifically focus on the users of the system and who has an advisory duty across the teams in project, as attempted in project-12. These “advisors” was of great importance as the organization in charge of project-12 had a strong emphasis on the value that was to be delivered and used. It was desired to have such advisors included in all the different teams, but the resources were too sparse to allow this. These advisors were normally hired by the customer. Additionally, the organization had a support-organization/management-organization outside the Scrum teams, consisting of 5 central resources, which worked across the different teams.

5.6.4 Feedback session: Scrum of Scrums and other meeting practices

In order to administrate and coordinate several teams working on the same project, it may prove beneficial to run daily Scrum of Scrum meetings, as seen in project-15. At first the project involved only a few teams, thus there was no need for Scrum of Scrums, but as time grew more teams got involved which lead to five parallel sprints being run. The daily Scrum of Scrum meetings lasted for 15 minutes and were arranged 30 minutes after the daily stand-up meeting, which was held in the morning. The rationale behind this practice being, that the *scrum masters* were informed of impediments during the daily stand-up meeting, although all these issues might not be possible to resolve within the team, such as impediments related to the infrastructure, the test environment or a missing delivery from an adjacent project that the team depended on. The solution was then for the *scrum masters* to take the identified impediments to the next level. The meetings were arranged on a daily basis to resolve issues that arose in the different teams as quick as possible in order to avoid having 2-3-4 teams waiting for, say, next week's Scrum of Scrums. Consequently, it was considered as the best solution to conduct meetings more frequently, while keeping them relatively short (15 minutes).

Scrum masters, and architects in some cases, attended these meetings. However, it was possible for a *scrum master* to bring one of the developers to the meeting in order to better communicate a problem or an impediment. It should be noted that some of the *scrum masters* had this role in more than one team. The reason being, that the work load was considered low enough to allow this arrangement.

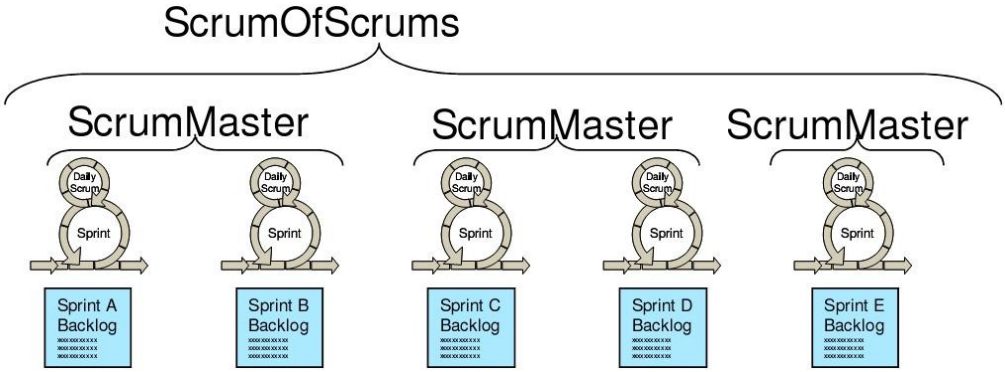

Figure 5.5: Scrum of Scrums.

When facing problems that cannot be solved within the project, but needs to be coordinated with other Scrum projects within the company, Meta Scrum[29] meetings may be arranged, as seen in project-15. During the meeting external problems were handled and various projects coordinated by synchronizing the program and focusing on common impediments. This was the most high-level kind of meeting in the organization. It allowed for knowledge sharing across projects through discussions of problems identified in the different projects and reuse of existing solutions. The Meta Scrums were arranged 1-2 times a week. The project manager for this project attended the Meta Scrums along with the customers, operators and people from production as well as other specialists within different fields. In some cases accountants, lead architects or other people attended, depending on the topic being discussed, although the management did not participate.

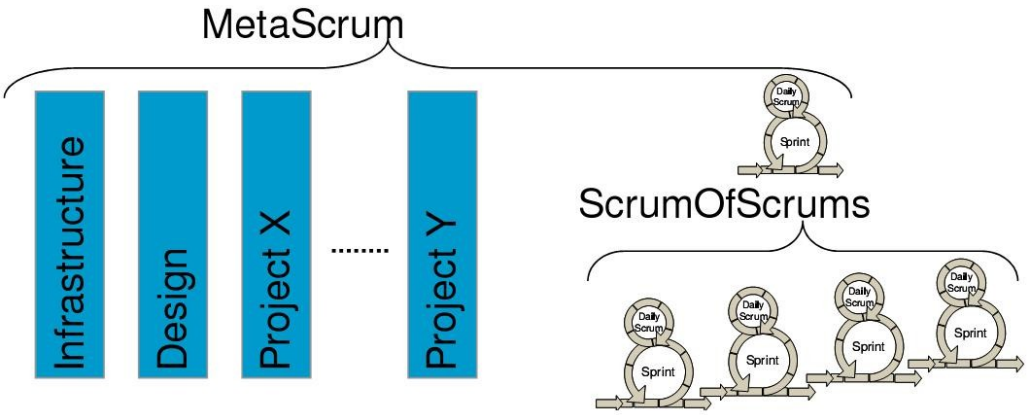

Figure 5.6: Meta Scrum.

In some cases Scrum of Scrum meetings has turned out to be a failure, thus an appropriate alternative might be to arrange a meeting with all the *scrum masters* in order to coordinate the work across the different teams better, as practiced in project-11. This meeting was held on a fortnightly basis and called governance meeting. During the governance meeting the discussions revolved around the work and typical problems related to the *scrum master* role, patterns for leading, the status in the different teams and ways to solve common issues and impediments the teams are experiencing as well as contracts, document and risk

[29] A Meta Scrum is high-level meeting that attempts to synchronize activities across the different projects within the organization. If an issue cannot be resolved within a project due to for instance dependencies related to other projects, it is typically handled in a Meta Scrum meeting.

handling. In addition, they also talked about the overall progress of the entire project, work related to the next release, ways to do Scrum and possible improvements to the process.

At first in project-11, however, weekly Scrum of Scrum meetings were arranged whereas one developer from each team attended along with all the *scrum masters*. The main reason to why these meetings were seen as a waste of time and thus discontinued, as pointed out by one of the *scrum masters*, could be that the developers attending the meeting were changed every week. This was done in order to foster knowledge sharing, however it lead to confusion and communication problems. Since there was no continuity regarding the people attending the meetings, issues brought up were never followed through.

Problems related to the Scrum of Scrums were reported from project-12 as well, even though it has been arranged three times a week up till now. However, increasingly fewer people attended these meetings as they appeared to include too broad a spectrum of teams to be of interest and consequently seen as a waste of time. A suggested solution has been to arrange smaller meetings involving 2-3 teams who are working on somewhat similar tasks and functionality. The rationale being that these teams have enough interests in common for the meetings to work out. If this turns out to hold true, the meetings will probably replace the Scrum of Scrums in the near future. Another proposed remedy to the Scrum of Scrums, is to make the teams as independent of each other as possible, to avoid coordinating activities through such meetings. In order for this to work, required communication between teams should be carried out through other more natural channels. This has been successful with two teams (in project-12) that are co-located. As a consequence, a desired situation would be to co-locate teams working on similar functionality, thus having common interests, which in turn would lead to a more natural increase in communication across the teams.

Another initiative to gain insight about status and the progress in other teams, and thereby enable mutual performance monitoring across teams, is to have *scrum masters* and people in similar positions attending other team's stand-up meetings as spectators from time to time, as reported from project-12.

In order to reward the team members, motivate them and encourage continued good work a sprint celebration may be added to the Scrum as an additional meeting, similar to what was practiced in project-12. The sprint celebration was held on the same day as the review and

the sprint retrospective and typically involved coffee, cookies and maybe a picture show displaying photos taken during the sprint. It was also customary to end the meeting by eating at a restaurant.

6 Discussion

This chapter starts by discussing the results from the literature (chapter 4) and the empirical studies (chapter 5) in regards with my problem definition, by structuring the discussion according to the behavioral makers of team leadership, as seen in chapter 4 and 5. Further on, section 6.1 identifies the threats to the validity of this report.

6.1 Overview

Due to project management being a very hard discipline to master and the deficient literature addressing this knowledge area in regards with Scrum, this report was motivated to answer the question: "What characterizes project management in mature Scrum teams?" through a literature study and an empirically supported analysis of Scrum teams in today's software development industry. Further on, the studied projects involve teams of varying maturity for comparison reasons, in which study-1,-2,-3 and project 1 in study-4 as well as project-11,-12,-13,-14 and -15 are considered to consist of mature teams. In particular, I regard project-11 as involving teams the with highest maturity-level, based on the fact that they have been working on the same product while running Scrum for 2.5 years and made a lot of well grounded adaptations along the way.

As seen from table 4.1 in section 4.1, 70% of the projects from the literature study were categorized as small or medium lasting for 1.5 year or less and involved somewhere around 5–6 software developers, while the rest (large projects) have a project duration ranging from about 1.5–3 years. Based on the selected studies, this indicates that most is currently known about how Scrum is performed in rather small teams. As a consequence, it is difficult to determine whether additional practices would have to be added in order to complement Scrum when used in a large scale project environment. Although my findings indicate that large (considering project length and size of teams) projects tend to involve more stakeholders, often in a distributed environment, which requires stronger customer involvement and on-site *scrum masters* and *product owners*. In addition, Scrum of Scrum initiatives and other meeting practices seem to be required to coordinate the various teams. Communication facilities supporting cooperation across geographically spread teams, such as teleconferencing and application sharing, may also be needed. Further on, as the teams studied in the literature in most cases had not been practicing Scrum for more than about a year, this confirms my persuasion that there is a need to study mature Scrum teams.

The increased diversity in team problem solving faced today is highly visible especially in large projects as seen in study-3 and project-11, where teams had to overcome both geographic and cultural boundaries when solving problems related to the project. This clearly imposes new challenges with regard to appropriate interaction patterns for information sharing, synchronization and combination of team member contributions and the facilitation of meetings which are all part of team leadership as seen from (Salas et al., 2005). However, team problem solving may be considered the most prominent challenge in a distributed project environment. The reason being that team problem solving usually requires a high-degree of face-to-face discussion, when moving around tangible objects such as task cards or explaining by writing or drawing, or in some cases working in pairs through e.g. pair programming.

6.2 Facilitate Team Problem Solving

In the literature, it was reported that an open office space often (study-4 and -5) facilitated more discussions among the team members, thus leading to increased team problem solving. Although, it was pointed out (in study-4) that being situated like this made it hard to focus. This was confirmed by project-16, which reported that an open office space lead to more frequent interruptions from people working on other projects that requested help. Such requests were often directed to the most experienced team members. The external inquires were found to disrupt team stability and the ability to establish shared mental model and coordination, and due to the constant interruptions tasks were often not completed, thus reducing the collective team motivation. As a result, facilitation of team problem solving was impaired, as reported in study-8 and project-11,-13,-14 and -16. In study-5, on the other hand, the team members were afraid of taking on more responsibility as customer support from previous areas of responsibility kept interrupting their work. However, no appropriate solution to this problem has been identified in the examined studies from the literature.

As Scrum does not include any project management mechanisms for appropriate handling of such external requests, the attempted solution employed in project-13 and -16 was to direct all requests through the *scrum master* and only handle them within an allotted time space. In project-13, a “service window” defining the amount of time to be used on request per day/week was established and within project-16 the *scrum master* gathered up all the

small inquiries until they amounted to one or two full days and then released resourced within the team to carry them out. These practices ensured that the impact of the inquiries were restricted which prevented them from pervading the actual work related to the project. Additionally, the time devoted to handling the inquiries could be scheduled in accordance with the project tasks.

A perhaps even better way of coping with the problem of external inquiries as it allows for knowledge accumulation within the team as well as the feeling of being trusted and having responsibility, which may be motivating factors in themselves, is the "Man-on-duty"-practice[30]: The members of the team swap on functioning as a gatekeeper for the team in regards to external inquiries during a sprint, in order for them to get the time to focus on the sprint tasks, which in turn allows tasks to be completed, thus boosting collective motivation to reach sprint as well as project goals. Each team member has this position for two days in a row. It allows everyone in the team to try to resolve the inquiry themselves or alternatively determine if it could be directed to another team or, as a last resort, release resources within the team to solve the task.

In order to facilitate a more effective sprint planning and the creation of a more realistic sprint backlog, newly employed developers or team members new to Scrum may observe the estimation process, during sprint planning, instead of partaking at first. During this phase, the more experienced team members estimate all tasks, and as the the novice team members complete tasks, they may develop a feel of the deviation between estimated and actual time used, which would enable them to participate more actively in the estimation process as time goes by. The reason to this practice is that neither the team, nor the novice team members will benefit from letting unexperience individuals estimate tasks at first, as much time may be wasted on estimation, while other team members with more experience perform the estimation better, as reported in project-16. In this project, the estimation was left to only a few team members due to lack of understanding, within the team, of the tasks to be carried out. Further on, after the estimation process is complete, it is advantageous to work on each user story in groups, if possible, as this allows for a

[30] A practice explained by an Acando representative during the brief speech „Scrum and Bugs" in the Agile Development Conference in Trondheim 2009.

higher degree of completion, which leads to a more lean process through less waste in the form of tasks in a commenced state.

In general, Scrum facilitates team problem solving to some degree by having the *scrum master* remove impediments and discussions during the sprint planning meetings, however, additional practices seems to be needed.

6.3 Provide Performance Expectations and Acceptable Interaction Patterns

Pair programming is not a part of Scrum, but it can be an advantageous interaction pattern for supporting the Scrum process as it allows for mutual performance monitoring, facilitates backup behavior and knowledge sharing as seen from project-11 and project-16 and from the literature in study-2 and -10. It was identified in the literature, in study-2, that pair programming can lead to a smaller code base, thus more efficient resources usage, through i.e. less debugging at later stages, since the defect rate is directly correlated with the code length. Pair programming may also be particularly suitable for difficult tasks, as indicated by study-10 and project-16. Furthermore, project-11 and -16 reported that pair programming can be useful for bringing new team members up to speed, teaching new practices (such as test-driven development) or simply to increase the overall competence in the team by having experienced members pair program with the more novice. However, it should be noted that pair programming can be perceived as a somewhat strenuous interaction pattern as people tend to have different work habits, as reported by one of the *scrum masters* in project-11.

The importance and thus the need for proper training for everyone involved (the team, *scrum master*, *product owner*) seems to become apparent as the organization matures. However, little in the literature is devoted to Scrum training, but project-11 reported that the certification of *scrum masters* in the project's remote office (in Malaysia) ensured a much better understanding of the Scrum process and higher work efficiency by allowing local *scrum masters* facilitate the team. The importance of the *product owner*-role also seems to be underrated, as seen in project-11 where the *product owners* did not receive proper training until after running Scrum for nearly 2.5 years. Further on, if working on a long-lived project it may be beneficial to hire a consultant company to review the Scrum process with focus on how the roles are working and how the meetings are run and give feedback and training in practices when found needed, as reported in project-11. An

alternative may be to have people working on another project, within the same organization, present what worked and what did not work out for them in regards with the Scrum process in order to pick up good practices, refresh concepts and increase awareness about caveats, as seen in project-16.

An easy, but maybe not so accurate way of maintaining the burndown chart is to simply draw it on a whiteboard, as seen in project-11. However, it may be more accurate to have the team members record hours spent on the project and remaining hours for tasks in progress on a shared spreadsheet, as reported to be common practice in project-13,-14 and -15. Although, this may be considered a cumbersome and strenuous process, which may also easily be forgotten, thus leading to a burndown chart that is not up-to-date. Consequently, a solution may be to alleviate some of the work involved in this process by arranging for mechanisms that allow for easy reporting of hours spent on tasks in the project, as seen in project-16. In this project, the team had a database where hours reported to be spent on the project in the payment system were automatically exported into the system for recording remaining hours and calculating the burndown chart. This practice allowed them to have only one place to record working hours thereby avoiding problems related to double booking, which saved the team valuable time. Consequently, the team members were encouraged the employees to update frequently, thus keeping the burndown chart closer to the actual values.

Nothing was reported in the literature about advanced performance expectation measurements such as team velocity. This may be due to the fact that the teams examined in the literature had not yet reached a Scrum maturity level were such metrics were needed or simply because the teams were not stable enough to calculate such metrics. However, it is apparent from project-11 and -13 that it is considered beneficial to calculate team velocity in the industry, although different practices for how this is done exist. In project-11, the term story points was introduced after the team made the transition from estimating tasks, during the sprint planning, in days to story points. As a consequence, story points produced during a sprint was used as a metric for team velocity, whereas one of the *scrum masters* remarked that his team was currently able to produce 0.75 story points per day, per person. In contrast, hours were used when estimating in project-13, thus hours was the metric employed when estimating team velocity. After the sprint, the deviation between estimated and actual time spent on the sprint backlog gave an indication of what the team was

capable of producing, thus a suggestion of team velocity and how the estimates should be adjusted accordingly for the next sprint. The number of tasks completed was also examined in project-13, thus revealing another indication of the amount of work the team was able to produce during a sprint.

6.4 Synchronize and Combine Individual Team Member Contributions

Synchronizing and attaining an overview of team member contributions may be facilitated during the daily stand-up meeting next to a task board by moving around post-it notes describing tasks to illustrate progress in the project, as seen in study-2 and project-11. This practice also enables mutual performance monitoring and coordination across the team. A similar practice was reported in project-16 using a computer, whereas the changes to the sprint tasks' status were demonstrated by moving tasks around on a digital task board displayed by a projector. However, the consequence was that the overview of the individual team member contributions was not visible once the meeting was over and the projector was turned off. Instead the team members had to visit a website to access this information. Compared to the approach from project-11, this practice reduced the availability and tangibility of the information, which may indicate that a hands-on approach would be more appropriate. Furthermore, the status updates and overview of progress (through moving the tasks) may be facilitated more effectively by having the team members prepare notes (maybe in the form of post-it notes) before the meeting, as reported in study-2.

The project status after synchronizing and combining individual team member contributions may be more easily communicated to the customers and higher management by involving the *product owner* and maybe a project manager[31] in the daily stand-up meetings, as seen in study-8. This practice may also clear up issues faster and allow customers to stay informed whilst issues happen as they stay in the loop, which will enable them to have a better idea when to expect questions, as reported in study-1. Additionally, it may be beneficial to include technical personnel to answer questions during the meeting, as

[31] The project manager-role is not a part of Scrum, but often included in Scrum projects in the industry to handle elements not addressed by Scrum such as budgeting.

seen in project-13 where an individual responsible for infrastructure attended. However, by having managers or others in positions of perceived authority attend the daily stand-up, this may produce a feeling, within the team, of being monitored. The result may be that the team members feel pressured to report major progress every day (even though this is not necessarily true) and can inhibit problems from being reported, thus reducing team self-management and encouraging micromanagement.

Individual team member contributions may also be synchronized within the team by including in the definition of done that each time a module is to be delivered a peer review must be performed. The peer review may also be carried out by having a co-worker from another team walk through and approve a checklist of everything from code standards to use of framework, which provides synchronization of contributions between teams, as seen in project-15.

The need to introduce test-driven development and automated testing to complement the Scrum process seems to be one of the first revelations for a maturing Scrum team, as seen from project-11,-12,-15,-16. This creates fertile conditions for sophisticated continuous integration schemes. Although not part of Scrum, continous integration can support Scrum by allowing the team to synchronize and combine individual team member contributions so that the current state of the product can be inspected, as also seen from the literature in study-4 (which was the only study from the literature that reported on the use of continuous integration). The continous integration schemes seem to grow more ambitious and automated as the organization matures. Furthermore, being able to deploy the latest code to a test environment on a regular basis enables customer feedback to be attained in a timelier manner, which greatly increases the chances of meeting the customer needs and wants, thus achieving customer satisfaction. Test-driven development combined with continuous integration also encourages code base improvements as team members are alleviated from hazards attended with changes that introduce new errors and affect other parts of the code, since the consequences of any changes will be picked up by the tests. Furthermore, initiatives may be employed to ensure that team members spend more time inspecting their code for errors before checking in, thus improving code quality, as seen in project-15. In this project, whenever someone checked in code that could not build, a red light was turned on automatically and a picture of the individual that checked in the code was displayed on a big screen. Additionally, by constantly monitoring how a project

progress, increased awareness can be realized in regards to potentially needed changes, which may lead to more agile team dynamics.

6.5 Seek and Evaluate Information that Affects Team Functioning

It has been identified through both the literature (study-2,-4,-8 and -10) and the industry (project-11 and -15) that a good practice for seeking and evaluating information that may affect team functioning is systematic use of a physical task board close to the team. By moving around tasks, represented by post-it notes, on the task board as progress is made valuable information is presented to the team members and actions can be taken accordingly. This practice can be extended by using pictures of the team members to indicate who are working on what at any given time, drawings indicating team members working on-site at the customer and placing all user stories involved in the sprint on the task board, as seen in project-11. Task-notes of different sizes can also be used to indicate the work load of each task, as reported from the literature in study-2. This can also be complemented by flipping task-notes upside down whenever a task is changed in order to make it easier for the *scrum master* to follow up the changes, as practiced in study-5. A further practice can be to dedicate a separate section of the task board for issues indicated by post-it notes, in a distinct color, with the number of hours spent trying to solve this issue, as seen in project-15. The hours spent trying to solve the issue will provide the *scrum master* with an indication of how much time that have been wasted on non-production related work.

Information affecting team functioning can also be provided through a product vision unique to the project, as seen project-12. This vision can be developed through a technique called “press release form the future”, which helps the team to define how a successful project would look like. This was common practice in project-12. Furthermore, the desire to include a vision to every Scrum project in the organization in charge of project-12, is largely congruent with the principles of leadership as defined by Kotter (Kotter, 1990). According to Kotter, leadership is about producing change and movement by establishing direction through creating a vision, clarifying the big picture and setting strategies. Furthermore, Kotter suggests that senior management should communicate the vision in order to give employees relevant timely information and solicit their feedback. This is further backed up by study-5 and -8, which reported on the need for communicating and presenting a proper product vision.

The product backlog is one of the most valuable information sources to the team and by keeping it readily accessible, increased transparency throughout the organization can be attained, thus making information which may affect team functioning more available, as advocated in study-8. Furthermore, fleshing out user stories is according to Mike Cohn (Cohn, 2004) an effective and appropriate technique for the team when evaluating information found in the product backlog, and was used to a large extent in project-11. As a result of this kind of process, the team members may become better informed with regards to how the current sprint backlog will affect team functioning.

The need to decrease the sprint length seems to arise as the team matures and gains better understanding for the Scrum process and thus is capable of administrating a shorter sprint, as reported in project-11, where the sprint length was changed from four weeks to two weeks. Furthermore, it was reported from both project-13 and -16 that it would have been advantageous to have shorter sprints as it would have made it possible to postpone handling some of the external requests until after the sprint. Shorter sprints with smaller goals, that is a shorter planning horizon, are also perceived as more motivating for the team and make it easier to stay agile in regards with unforeseen changes. It is also easier to remember issues arising during the sprint when attending the next sprint retrospective, which is essential since issues have to be brought up in order to be solved. However, it is important to adjust the sprint planning according to the sprint length in order to avoid overhead. This could be done by devoting one hour pr. week to planning, as seen in project-11.

It seems that the use of software tools such as discussion forums, wikis and blogs to developer manuals for seeking and evaluating information that may affect team functioning increase as the Scrum teams mature. These sources of information usually contains the expected code-style, how to commit your code, information about test-driven development, what tools to use, links for how to use the tools and so on. This way of distributing information throughout the team and to management is considered more appropriate than the use of mailing lists to send status reports, as seen from the literature in study-4. The reason being that it is easier to distribute both text and images and there is no risk of information being caught in the spam filter or getting lost among large amounts of mails. Wikis was used in project-11 at first and then blogs were introduced instead as the threshold to enter new information into a blog was perceived to be lower, which would make the information

source grow at a larger rate. A discussion forum was also reported to have a positive effect on training and knowledge sharing in project-16 and is recommended in the Pragmatic Programmer (Hunt & Thomas, 1999). Having both technical books and books about the process such as books describing how to write user stories available to the team have been reported to have a positive effect in both large projects (such as project-11) and small projects (such as project-16). Furthermore, the *scrum master* can provide reports and research studies, in order for the team to stay up-to-date and be able to continuously inspect and adapt the Scrum process to stay agile.

6.6 Engage in Preparatory Meetings and Feedback Sessions with the Team

It seems to be a trend that as a team becomes more mature using Scrum, parallel processes for pre-planning the next sprint will be added to the Scrum framework in order to make the sprint planning more effective, as seen in project-11 and desired in project-12. In project-11, the pre-planning phase involved the *product owner* sending a prioritized list of items from the backlog to each team member during a sprint, who then considered the task at hand, how to break it up and made some initial estimates for each task and sent it back to the *product owner*. Based on the feedback from the pre-planning phase, the *product owner* usually made a draft which the team used as a starting point when planning the next sprint. The team members' participation prior to the sprint planning also ensured that the team members were highly prepared for the meeting. An alternative approach may be to have the *scrum master*, *product owner* and project manager arranged a separate meeting to prepare the product backlog for the next sprint, as seen in the literature from study-9. Another similar initiative may be to let the *scrum master* actively start planning and creating long term visions ahead of time with the customer, to assist them formulate their ideas with enough detail for the developers to employ them when estimating, as seen in study-1. This may ensure that the *scrum master* and the customers are better prepared for the actual planning meetings, thus making them easier to facilitate and more effective.

It appears that the literature does not provide much detail about how to run the review and retrospective meetings, while several interesting practices may be extracted from the findings in the industry. However, if possible, it may be beneficial to involve top management in the review meetings, as seen in study-4, suggesting easier tracking of project progress and a more agile project as information flow more freely through the organization allowing

changes to be acted upon faster. Additionally, a useful practice to employ during the retrospective meeting may be to let each team member explain how they experienced the last sprint, while illustrating this through an energy curve on a whiteboard, as seen in project-11. The reason being, that this has a psychological effect on the human mind, causing a person not to lie when drawing the curve (Derby & Larsen, 2006). Further on, a structured way of presenting feedback from the team members is to let them write down what went well on, say, green post-it notes while using red/pink ones for negative feedback, and place them on a wall one-by-one while explaining each note, as practiced in project-11. The color coding of the post-it notes makes it possible to quickly identify the sign of a given response and to get an overview of the sign allocation of the feedback. This feedback session often leads to discussion among the team members which may facilitate team orientation and shared mental models. Afterwards, the feedback-notes should be left hanging in near vicinity of the team to remind them of following the issues through, as suggested in by one of the developers in project-16.

Recently, in project-12, they introduced the concept of retrospectives of retrospective meetings in order to better coordinate teams and facilitate knowledge sharing. Initially, this is carried out with two teams who work closely together. Each team first have their sprint retrospective separately and then join forces to have a retrospective meeting across the two teams. Another somewhat similar alternative may be to let *scrum masters* facilitate other teams' retrospectives, in order to enable knowledge sharing across teams.

In large projects involving many teams it may be beneficial to have a group of people that work across the teams and that offers advice and discuss important issues, that the *product owner* cannot solve by himself, which may impact all teams, as used in project-11 and -12.

From the literature, only study-3 reported on the use of Scrum of Scrums, while based on the findings from the industry it seems that the Scrum of Scrums created mixed feelings regarding its purpose and effect in several projects. It has been identified that several companies tend to substitute this meeting with workings of their own to suit the organization's needs, as seen in large projects such as -11,-12 and -15. This indicates increased ability to stay agile and adapt as the organization matures with time.

In order to administrate and coordinate several teams working on the same project, it may prove beneficial to run daily Scrum of Scrum meetings for the *scrum masters*, as seen in

project-15. By arranging the meeting after the daily stand-up, the *scrum masters* will be informed of any impediments that might not be possible to resolve within the team (such as impediments related to the infrastructure), in order for these to be handled during the Scrum of Scrums. These meetings may also facilitate discussions regarding the work and typical problems related to the *scrum master* role, patterns for leading, the status in the different teams and ways to solve common issues and impediments the teams are experiencing as well as document and risk handling. When facing problems that cannot be solved within the project, but needs to be coordinated with other projects within the company, Meta Scrum meetings may be arranged, as seen in project-15. However, in some projects the Scrum of Scrums may turn out to be a failure due to e.g. lack of common interest in the topics discussed or issues not being followed through due to deficient continuity regarding the people attending the meetings. In such projects, an appropriate alternative might be to arrange smaller meetings involving 2-3 teams who are working on somewhat similar tasks and functionality or to simply co-locate them to facilitate inter-team communication, as suggested in project-12. Yet another solution is to have the *scrum master* or people in similar positions attend other team's stand-up meetings to gain insight about status and project progress, as advised by the chief architect in project-12.

In order to reward the team members, motivate them and encourage continued good work a sprint celebration may be added to the Scrum as an additional meeting, similar to the practice in project-12.

A problem often faced is that meetings in general tend to drag on. Based on practices found in study-1 and -9 from the literature and project-11 and -16 in the industry, it seems that in order to keep meetings short and focused, the participants need to be as prepared as possible.

6.7 Indications of General Trends Identified

To sustain continuous improvement in large Scrum projects with multiple teams when the teams are maturing, it is important to create incentives (not necessarily monetary) and to prepare for ways of integrating the solo specialists into a cross-functional team in order to foster self-organization within the team, as indicated from the discussion in study-8. The need to solve problems related to self-organization within teams consisting of specialists,

were also reported from project-13. Additionally, study-8 reports that management should be motivated to prioritize decisions on a regular basis.

In order to be successful with Scrum it is important that also the *scrum master* matures along with the team and the process itself. Based on the *scrum master*-role's focus on facilitating the Scrum process for the team by giving individual guidance and assisting the team members by removing impediments to their work, there is reason to believe that situational leadership (transitioning between directing, coaching, supporting and delegating depending on the situation) should be employed by a mature *scrum master*. This is further backed up by looking at how (Blanchard, Zigarmi, & Zigarmi, 1985) defines the concept; "Situational leadership stresses that leaders need to find out about their subordinates' needs and then adapt their style accordingly. Leaders cannot lead using a single style; they must be willing to change their style to meet the requirements of the situation." However, it should be noted that the situational leadership model has also received a lot of criticism.

The role of the teams in Scrum closely resembles the understanding gained from the (Norwegian) project by Thorsrud and Emery (Thorsrud & Emery, 1970), whereas cooperation attempts between LO and NHO formed partly self-organizing workgroups which initiated innovative thinking regarding these kinds of work groups. It is claimed in "Arbeid i Team" (Levin & Rolfsen, 2004) that this is an important fundament to shaping the ideas around teams. Sørhaug (Sørhaug, 1996), takes this a step further by stating that leadership in these self-organizing workgroups must be based on other premises than exercising authority and the use of power. The way to practice leadership, in these workgroups, should rather have a form that contributed to motivation, involvement, development and maintenance of the fellowship. Flexible leadership and a high degree of conformity became two of the foundation pillars in the partly self-organizing groups. These are values recognizable from how the Scrum teams are run, where the *scrum master* acts like a coach that merely assists the team while the team itself resolves problems regarding their tasks.

As some teams tries to cooperate to complete the user stories chosen for a sprint in order to eliminate waste in the form of half-finished tasks, perhaps making the whole organization lean is the next step for a company consisting of mature Scrum teams, as indicated by one of the *scrum masters* in project-11. This philosophy is more holistic and revolves around increasing the efficiency of the organization's product chain and performing analysis to

identify bottle necks. Lean pervades the whole organization and is typically introduced in a top-down manner which involves commitment from top management, whereas the focus is on eliminating everything that produce waste, such as unnecessary documentation or already started tasks that are not completed. However, this is a big step and requires a high-level of maturity, insight and dedication.

6.8 Threats to Validity

The threats to the validity of the results in this report that has been identified are:

- A low number (1-3) of key personnel were interviewed in regards with each project, which may prevent all relevant information about each project to be un-covered. Additionally, this may give a rather biased view of the project.

- It can be argued that sixteen projects ranging from small to large, comprises a too insignificant fraction of projects from each section to be able to say something about projects of the different sizes in general. However, according to (Yin, 2002) even a single case study allows for theoretical generalizations to be made.

- The choice of framework, team leadership from the "Big Five" of teamwork (Salas et al., 2005), employed to characterize the practices used in various Scrum teams is not necessarily the most appropriate one. Some of the behavioral makers were found to be a bit overlapping, thus resulting in aspects of Scrum being examined and discussed in more than one place; the practice of using a task board is both applicable to "provide performance expectations and acceptable interaction patterns" and "seek and evaluate information that affects team functioning." Sprint planning can be discussed both under "facilitate team problem solving" and "engage in preparatory meetings and feedback sessions". Furthermore, daily stand-up and review meetings are relevant to both "synchronize and combine individual team member contributions" and "engage in preparatory meetings and feedback sessions". Additionally, it was necessary to remove one of the original behavioral makers of team leadership, namely "clarify team member roles." The reason being that Scrum considers the team as a role in itself, since the team members' areas of responsibility will vary both within a sprint and across sprints due the team being self-organizing and cross-functional. Further on, given the time constraints imposed on this master thesis, there was no time to carry out a comprehensive analysis of potential frameworks, thus any of the frameworks of teamwork behavior listed by (Rousseau, Aube, & Savoie, 2006), or others might have been more appropriate. However, I consider the chosen framework satisfactory to characterize Scrum practices and recommend it for other research projects whereas team leadership is being investigated.

7 Conclusion

This report has investigated what characterizes mature Scrum teams. The behavioral makers of team leadership from (Salas et al., 2005) was used as a framework to identify and characterize important practices employed by the teams in the projects examined. Further on, based on multiple studies, I have found that several practices must complement Scrum to achieve successful project management. The most prominent practices found to be typical for mature Scrum teams are summarized below and I believe that the results obtained from this report can be applicable to other Scrum projects in the industry today.

Facilitate team problem solving

- As external requests interrupt the facilitation of team problem solving, they can be directed to the *scrum master* and only handled when the amount of work to be carried out corresponds to the established service window. Alternatively, a man-on-duty practice can be employed, where the team members swaps on gate keeping external requests, which accumulates team knowledge, gives the team members responsibility, the feeling of being trusted and increases motivation.

- In order to achieve a more effective sprint planning, team members not familiar with Scrum should participate in team problem solving when breaking up the work into tasks, although only observe the estimation process at first. Moreover, the more experienced team members should estimate the tasks until the novice developers have established a feel for how long it takes to complete tasks.

Provide performance expectations and acceptable interaction patterns

- Pair programming can be an advantageous interaction pattern for supporting the Scrum process as it allows for mutual performance monitoring, facilitates backup behavior, knowledge sharing and leads to a smaller code base. It is an especially useful interaction pattern for peer reviewing code, increasing overall competence in teams by bringing new team members up to speed and teaching new practices.

- The *scrum masters* and *product owners* should receive proper training to ensure that they fully comprehend their role in Scrum. Training of all parts must be followed up, as it is easy to fall back to old habits after the introduction to Scrum. Consequently, a consultancy company may be hired to refresh Scrum concepts and improve the process by reviewing how Scrum is run. Alternatively, people working in other projects may present their experiences with Scrum to the team.

- To save time, avoid double booking and to keep the burndown chart closer to real values, hours reported being used on the project should be automatically exported from the payroll system to the system maintaining the burndown chart.

- Utilize the same metric as used when estimating, hours/story points, to calculate team velocity. Additionally, the number of tasks completed in a sprint may be tracked.

Synchronize and combine individual team member contributions

- In addition to the daily stand-up and the review meeting, it may be advantageous to complement Scrum in order to synchronize and combine individual team member contributions through peer reviewing of code, test-driven development, automated testing and continuous integration schemes. By deploying the latest code to a test environment on a regular basis, the current state of the product can be inspected thus enabling customer feedback to be attained in a timelier manner.

- Status updates and overview of progress when moving notes describing tasks, during the daily stand-up, may be facilitated more effectively by having the team members prepare notes (with issues etc.) before the meeting. This will also make the meetings more interesting for the team members, as they are better prepared.

Seek and evaluate information that affects team functioning

- A good practice is to use a physical task board close to the team with pictures of the team members attached to the appurtenant tasks and drawings indicating who is working on-site at the customer. Task notes of different sizes can also be used to indicate the work load of each task and task notes may be flipped upside down whenever a task is changed, while all user stories involved in the sprint can be placed on the task board for increased visibility.During sprints, a separate section of the task board may be kept for issues, indicated by post-it notes in a distinct color, with the number of hours spent trying to solve this issue, to illustrate time wasted.

- As shared mental models are important, a product vision may be developed to help the team define how a successful project would look like. By having the team flesh out user stories from the product backlog, they will be better informed with regards to how the current sprint backlog will affect team functioning. Decrease the sprint length, as shorter sprints with smaller goals are perceived as more motivating and feedback on teamwork can be attained more often, thus making it easier to adapt to change. It also makes it easier to remember issues arising during the sprint in the retrospectives. Utilize blogs as developer manuals for training and knowledge sharing since they are easy to maintain and search. Additionally, technical books and books about the process should be available to the team.

Engage in preparatory meetings and feedback sessions with the team

- A pre-planning phase may be employed during the current sprint in order to plan the upcoming one, thus making the sprint planning easier to facilitate, more effective and allows the participants to be better prepared.

- During the retrospective, each team member may explain how they experienced the last sprint by drawing an energy curve on e.g. a whiteboard, because drawing the curve makes it hard not to be truthful. In order to easily indicate the sign of the feedback, what went well may be written down on green post-it notes while using red ones for negative feedback. The feedback-notes should be left hanging in near vicinity of the team to remind them of the identified areas for improvement.

- In order to better coordinate teams and facilitate knowledge sharing, retrospective of retrospective meetings could be arranged or *scrum masters* could facilitate other teams' retrospectives. Scrum of Scrums tend to create mixed feelings and may be substituted with other practices that suits the project, such as running a daily meeting involving all *scrum masters*, for 15 minutes, after the daily stand-up.

General conclusions

In order to improve team leadership and foster self-organization within the team, solo specialists should be integrated through e.g. involvement in discussions and team problem solving to form cross-functional teams, while *scrum masters* must be willing to change their leading style to meet the requirements of the situation. Further on, the team should cooperate by e.g. working on the same user story to reduce the number of commenced tasks, thus motivating the team by completing additional tasks and making the project leaner. This improves team leadership by facilitating team problem solving and making it easier to synchronize and combine individual team member contributions as the tasks undertaken are interrelated.

7.1 Implications for Research and Practice

The findings in this study will hopefully allow companies in the early phases of Scrum to adapt and improve their process as well as guide maturing Scrum teams by highlighting practices that may help them manage projects more successfully. Future studies should seek to confirm these findings in additional projects consisting of mature Scrum teams. Furthermore, future research avenues arising from this report involve:

- It is relevant to the facilitation of team problem solving, within team leadership, to investigate what the *scrum master*-role involve and how *scrum masters* should remove impediments for the team members, assist and lead them.

- What the *product owner*-role involve should be investigated, as this role is vital to the prioritizing of project work, thus imperative to project management in Scrum.

- How can the problems and uncertainty related to the planning horizont be handled? Project management can be improved if the goals for the next 2 months are known.

- Shared mental models within a team are very important in regards with team leadership, thus it may be interesting to examine processes that facilitate this.

- As it may be hard to establish and communicate a clear definition of done, which is related to performance expectations within team leadership, the appropriateness of employing user stories to understand the work at hand and set up acceptance criteria should be examined.

8 References

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). *New Directions on Agile Methods: A Comparative Analysis*. Paper presented at the 25th International Conference on Software Engineering, ICSE'03.

Augustine, S., Payne, B., Sencindiver, F., & Woodcock, S. (2005). Agile Project Management: steering from the edges. *Communications of the ACM, 48*(12), p.85-89.

Bates, C., & Yates, S. (2008). *Scrum Down: A Software Engineer and a Sociologist Explore the implementation of an Agile Method*. Paper presented at the CHASE '08.

Beck, K., & Andres, C. (2004). *Extreme Prgramming Explained: Embrace Change* (2nd ed.). Reading: Addison-Wesley.

Benbasat, I., Goldstein, D., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly, 11*(3), 369-386.

Blanchard, K., Zigarmi, P., & Zigarmi, D. (1985). *Leadership and the One Minute Manager: Increasing Effectiveness Through Situational Leadership*. New York: William Morrow.

Chau, T., & Maurer, F. (2004). Knowledge Sharing in Agile Software Teams. In W. Lenski (Ed.), *Logic versus Approximation* (Vol. 3075, pp. pp. 173-183). Berlin: Springer-Verlag Heidelberg.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Boston, MA, USA: Addison-Wesley Professional.

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2009). *The Scrum Primer v.1.1*: Scrum Training Institute: www.ScrumTI.com.

Derby, E., & Larsen, D. (2006). *Agile Retrospectives: Making Good Teams Great*: Pragmatic Bookshelf.

Dingsøyr, T., Dybå, T., & Abrahamsson, P. (2008, 4-8.August). *A Preliminary Roadmap for Empirical Research on Agile Software Development.* Paper presented at the Agile Conference '08.

Dingsøyr, T., Hanssen, G. K., Dybå, T., Anker, G., & Nygaard, J. O. (2006). Developing Software with Scrum in a Small Cross-Organizational Project. In I. R. P. Runeson & R. Messnarz (Eds.), *Euro SPI 2006* (Vol. 4257, pp. 5-15): Springer-Verlag Berlin Heidelberg.

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

Elwer, P. (2008). Agile Project Development at Intel: A Scrum Odyssey. *Danube Case Study: Intel Corporation*, pp. 1-14.

Chapter 8 References

Fernandez, D. J., & Fernandez, J. D. (Winter 2008-2009). Agile project management - agilism versus traditional approaches. *Journal of Computer Information Systems, 49*(2), pp. 10-17.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200-213.

Forsberg, E., & Lindström, M. (2008). Scrum med XP-relaterade tekniker - Införandet av Scrum och dess påverkan på systemutvecklargruppen.

Freeland, R. F. (2000). Creating holdup through vertical integration: Fisher Body revisited. *Journal of Law & Economics*, p 33-66.

Halkola, L. (2008). R&D Process Optimization for a Customer and Order Management System. *Master thesis*.

Hasnain, E., & Hall, T. (2008). *Preliminary Investigation of Stand-up Meetings in Agile Methods*: International Association of Engineers.

Hawryszkiewycz, I. (2000). *Introduction to Systems Analysis and Design* (5 ed.): Prentice Hall.

Hayes, S., & Richardson, I. (2008). Scrum Implementation Using Kotter's Change Model. In P. Abrahamsson (Ed.), *XP 2008* (pp. 161-171). Berlin Heidelberg: Springer-Verlag.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Boston, MA, USA: Addison-Wesley.

Hunt, A., & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*: Addison-Wesley Professional.

Jewels, T. (2003). The Dag-Brücken ASRS Case Study. *Journal of Information Systems Education, 14*(3), p.247-257.

Karlström, D., & Runeson, P. (2006). Integrating Agile Software Development into Stage-Gate managed Product Development. *Empirical Software Engineering, 11*(2), p. 203-225.

Kniberg, H. (2007). *Scrum and XP from the Trenches*: C4Media.

Kotter, J. P. (1990). *A Force for Change: How Leadership Differs from Management*. New York: Free Press. pp. 3-8.

Levin, M., & Rolfsen, M. (2004). *Arbeid i Team – Læring og utvikling i team*. Oslo: Fagbokforlaget. Kap.14.

Mann, C., & Maurer, F. (2005). *A case study on the Impact of Scrum on Overtime and Customer Satisfaction.* Paper presented at the (in Proceedings of) Agile 2005, Denver.

Chapter 8 References

Marchenko, A., & Abrahamsson, P. (2008). *Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges*. Paper presented at the Agile Conference '08.

Mathieu, M. A. M. J. E., & Zaccaro, S. J. (2001). A temporally based framework and taxonomy of team processes. *Academy of Management Review, 26*(3), p. 356-376.

Maurer, F., & Melnik, G. (2006). *Agile Methods: Moving towards the Mainstream of the Software Industry*. Paper presented at the ICSE '06.

Maylor, H. (2001). Beyond the Gantt Chart: Project Management Moving on. *European Management Journal, 19*(1), p. 92-100.

McGregor, D. (1960). *The Human Side of Enterprise*: McGraw Hill Higher Education.

Moe, N. B., & Aurum, A. (2008). *Understanding Decision-Making in Agile Software Development: a Case-study*. Paper presented at the 34th Euromicro Conference Software Engineering and Advanced Applications, SEAA '08.

Moe, N. B., & Dingsøyr, T. (2008). Scrum and Team Effectiveness: Theory and Practice. In P. Abrahamsson (Ed.), *Agile Processes in Software Engineering and Extreme Programming* (Vol. 9, pp. 11-20). Berlin: Springer-Verlag Heidelberg.

Moe, N. B., Dingsøyr, T., & Dybå, T. (2008). *Understanding Self-Organizing Teams in Agile Software Development.* Paper presented at the 19th Australian Conference on Software Engineering (aswec 2008).

Moe, N. B., Dingsøyr, T., & Kvangardsnes, Ø. (2009). Understanding Shared Leadership in Agile Development: A Case Study.

Murray, C. J. (2008). Lean and Agile Software Development: A Case Study. *Master thesis, MIT*.

Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies. *Communications of the ACM, 50*(3), 79-83.

Parsons, D., Ryu, H., & Lal, R. (2007). The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects. In T. McMaster, D. Wastell, E. Ferneley & J. DeGross (Eds.), *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda* (Vol. 235, pp. 235-249). Boston: Springer.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices in communication in software development. In T. Dybå (Ed.), *Empirical Software Engineering* (pp. 303-337): Springer Science + Business Media, LLC.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008a). *Distributed Agile Development: Using Scrum in a Large Project*. Paper presented at the IEEE International Conference on Global Software Engineering, ICGSE '08.

Chapter 8 References

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008b). Using Scrum in a Globally Distributed Project: A Case Study. In *Software Process Improvement and Practice* (pp. 527-544): John Wiley & Sons, Ltd.

Rising, L., & Janoff, N. S. (2000). The Scrum Software Development Process for Small Teams. *IEEE Software, 17*(4), 26-32.

Rousseau, V., Aube, C., & Savoie, A. (2006). Teamwork Behaviors: A Review and an Integration of Frameworks. *Small Group Research, 37*(5), p. 540-570.

Royce, W. W. (1987). *Managing the development of large software systems: concepts and techiniques.* Paper presented at the ICSE '87: Proceedings of the 9th international conference on Software Engineering, Los Alamitos, CA, USA.

Salas, E., Sims, D. E., & Burke, C. S. (2005). Is There a "Big Five" in Teamwork. *Small Group Research 2005, 36*(5), p.555-599.

Salo, O., & Abrahamsson, P. (2006). An Iterative Improvement Process for Agile Software Development. In *Software Process Improvement and Practice* (pp. 81-100): Wiley InterScience.

Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations - a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET Software, 2*(1), pp. 58-64.

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Sharp, H., Robinson, H., & Petre, M. (2008). The role physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers*.

Sørensen, M. R., Jense, H. M., & Holm, M. (2008). Agil videnledelse med Scrum: en undersøgelse af dokumentationens rolle i systemudvikling. *Master thesis*.

Sørhaug, T. (1996). *Om ledelse: Makt og tillit i moderne organisering*. Oslo: Universitetsforlaget.

Tata, J., & Prasad, S. (2004). Team Self-management, Organizational Structure, and Judgements of Team Effectiveness. *Journal of Managerial Issues, 16*(2), 248-265.

Thorsrud, E., & Emery, F. E. (1970). *Mot en ny bedriftsorganisasjon*. Oslo: Universitetsforlaget.

Yin, R. X. (2002). *Case Study Research: Design and Methods* (Third ed. Vol. 5). Thousand Oaks, CA: Sage Publications.

A. Appendix - Interview Guide[32]

- *Hvor mange er dere i hvert team?*
- *Hvor lenge har hvert team jobbet sammen? Hva er typisk her og hvilke fordeler/ ulemper er det i forbindelse med utskiftning av team medlemmer?*
- *Hva er typisk sprint-lengde dere bruker? Er det noen spesiell grunn til dette valget? Har dere endret sprint lengde i løpet av den tiden dere har kjørt Scrum?*
- *Hvordan foregår et sprint planleggingsmøte hos dere? Hvordan estimeres arbeids-mengden som er nødvendig for å løse hver enkelt task eller user story?*
- *Hvordan fasiliterer dere team problem løsning?*
- *Har dere metoder for å måle prosjektfremgang (f.eks burndown chart) og har dere metoder for å holde oversikt over hvor mye teamet produserer i løpet av en sprint? (team velocity)*
- *Hvordan synkroniserer dere bidrag fra de ulike team medlemmene og hvilken rolle hadde scrum master i denne prosessen? (ble dette gjort i løpet av daily stand-up eller ble retningslinjer satt opp i planleggingsmøtet?)*
- *Hva slags informasjonskilder vil du si påvirket hvordan teamet jobbet på? Hvordan ble informasjon som var viktig for teamet funnet og hvem jobbet med å skaffe til veie slik informasjon? (product backlog, sprint backlog, user stories, story board, produktvisjon, blogg, e-post liste, wiki-side)*
- *Hva ser du på som din viktigste oppgave som utfra rollen din under en Scrum og hvilke tiltak gjør du for å oppnå dette?*
- *Praktiserer dere Scrum av Scrums møter? (i såfall) Hvem deltar og hva blir typisk tatt opp på slike møter? Hvor ofte foregår disse møtene?*
- *Praktiserer dere noen andre former for møter? (i såfall) Kan du fortelle litt om disse?*

[32] The interview guide has been continually improved throughout the interviews and this is the final version.

- *Hvorden lærer dere opp nye teammedlemmene og hvordan har dere gjort dette tidligere i prosjketet? Hvordan skjer vanligvis opplæringen av en scrum master og en product owner hos dere?*

- *Ser du noen utfordringer i forbindelse med prosjektledelse innenfor Scrum som vi ikke har vært innom (til nå)? Er det noe du synes er spesielt vanskelig å få til når man følger Scrum metodologien?*

- *Hvordan har deres bruk av Scrum endret seg etterhvert som tiden har gått og dere har blitt mer komfortable med å bruke Scrum?*

- *Hvilke grep har dere måttet gjøre for å tilpasse Scrum til bedriften? Har dere noen praksiser som ikke inngår i Scrum som må til for å få gjennomført prosjekter? (i såfall) Gjør disse praksisene dere mindre smidige?*

- *Hvordan foregår review- & retrospektivmøtene hos dere? Føler du at problemer som blir identifisert blir tatt tak i og endret til neste sprint slik at dere hele tiden er smidige eller føler du at man kanskje alltid får så mye ut av disee møtene?*

- *Kombinerer dere Scrum med XP-teknikker? I såfall hvilke? Er det noen faste teknikker som brukes eller varierer dette mellom sprinter eller mellom prosjekter? Brukes f.eks par-programmering hovedsaklig til å skrive kode eller til å gjennomgå/ review kode?*

- *Tar dere sikte på å oppnå continuous integration? (i såfall) Hvordan gjør dere dette? Hvordan opererer dere i forhold til testdrevet utvikling og automatisert testing?*

- *Hvilke mekanismer bruker dere i forbindelse med dokumentasjon? Hvordan er retningslinjene i forhold til dokumentasjons-mengde?*

- *Hvor smidige vil du si at dere er på det prosjeket du jobber på nå? Er det slik at dere klarer å tilpasse dere ved å gjøre nødvendige endringer underveis?*

- *Er det noe du vil trekke frem som du synes at dere har blitt mye bedre på i løpet av tiden dere har kjørt Scrum eller et aspekt ved Scrum du føler at dere får spesielt bra til?*

B. Appendix – Interviews

All the interviews were recorded and the sound recordings were transcribed whereas some sentences were re-written in order to make the text more readable. The questions are also written out in full length (the way they were intended to be) as the interviewee often started responding to a question before it was completed. All questions are written in italic and questions from the interview guide (at least the version of the interview guide at that time) are underlined and marked with bullet points. Furthermore, the interviews were conducted in Norwegian as the interviewees were from Norway. Consequently, all the transcribed material is also in Norwegian.

B.1 Interview with developer in project-11, 23.February 09

- *Hvor mange er dere typisk i hvert team?*

Det varierer litt. Det er egentlig alt etter hvor mange ressurser (vi har tilgjengelig). Det vi prøvde før jul for eksempel, var at vi hadde én fra DM-teamet (”Data Management”-teamet) og én fra Framework-teamet. Da var det DM som var prioritert, så de var fem på teamet, mens vi var to på teamet i EC-IS (EC er produktet organisasjonen utvikler). Det å være to på et team var litt lite. Det er en som er leid ut og så er det to stykker som har dratt ut (til kunden), så vi er egentlig et 5-manns team.

Normal størrelse på et team er vel rundt fire, fem eller seks stykker. Vi har hvertfall funnet ut at det er greit å være mer enn to.

- *Hvor lenge er det vanlig at et team har jobbet sammen? Jeg ser jo at det er en del utskiftninger...*

Ja, det har vi også hatt litt problemer rundt. Det er jo et problem dersom vi skifter ut (teammedlemmer) for mye, men samtidig så må man ha litt utskiftning for å få spredd kunnskap. Det er viktig å ha litt utskiftning, men ikke for mye. Jeg har vært i Framework-teamet siden jeg begynte her, og jeg har vært her i to år. Samtidig er jeg den eneste personen som har vært her (i dette teamet) i to år. Utvikler-1 (en annen utvikler) har vært her i teamet mesteparten av tiden, men han hadde en periode i høst hvor han jobbet i EC-DM. Så har vi utvikler-4, han er på vei ut nå. Han ble flyttet ut av teamet i (fjor) sommer, men var her (i Framework-teamet) frem til da.

Kommer du på noen spesielle fordeler ved å ha litt utskiftning av teammedlemmene?

Vi er veldig bygd opp mot moduler. Jeg kan gå i Framework-teamet eller EC-DM, så er det et Revenue-team, et Sales-team og til og med et Transport-team. Slik dette har foregått har man vært veldig isolert, nærmest som i sitt eget univers og vært veldig fokusert på hva man jobbet med. Man vet lite om hva som foregår i resten av verden (i de andre teamene) eller hva som skjer ellers. Dersom man har litt utskiftning, får man inn nye impulser. Noen kan si; "ja det har jo vi gjort og det kan du bruke, gjør dette isteden for" (lære av andre folks erfaringer vedrørende tilsvarende problemer og bruk av verktøy). Så slipper man å gjøre alt selv, og man får en del kunnskapsutveksling. Når man skifter mellom teamene, så må man passe på at man ikke skifter for mye for da kan man miste kompetanse.

- *Hva er typisk sprint-lengde som dere bruker?*

Vi hadde jo en måned (som standard sprint-lengde) ganske lenge. Backloggen kan skifte fort og da man f.eks få veldig mye support eller lignende. For å få det litt mer forutsigbart og større mulighet for å klare det (bli ferdig med alle oppgaven i sprint-loggen), så har vi satt ned sprint-lengden til 14 dager. Hvis det da kommer inn support (hendvendelser) så kan du, på en måte, si at dette må vente for vi skal ha planleggingsmøte på fredag. Så vi kan gi dem mer nøyaktige estimat på når de kan forvente svar da. Noe må du ta med en gang, men da kan man lettere planlegge dette inn i sprinten. Det (kortere sprint-lengde) gjør det lettere å tilpasse (uventede) ting. Det er en stor fordel, og er noe vi har begynt med etter jul. Dette har vi kjørt over 3-4 sprinter.

Hvordan syns du det har fungert frem til nå?

Jeg syns det fungerer veldig bra. Det er klart at det er et problem at det går mye tid rundt planleggingen. Da må man strømlinjeforme det litt, slik at det ikke går for mye tid vekk i dill. Jeg syns det fungerer veldig greit. Vi hadde litt problemer tidligere, da hadde vi en ting vi skulle planlegge, og så skulle vi gjøre det (utføre det i praksis). Når du da skulle planlegge en måned frem i tid, og allerede på dag nr to ser at dette går "råkt at skogen", så ble det deprimerende. Det er mye bedre når du setter deg mindre mål som er mer oversiktlige så gjør du dem, og så får du nye (setter du deg nye mål).

- *Bruker dere noen spesielle XP-teknikker som for eksempel par programmering?*

Vi har brukt en del par programmering, men jeg har ikke deltatt på de siste sprintene, jeg sitter litt på utsiden jeg driver helst med support. Jeg vet hvertfall at utvikler-1 (en av utviklerne i teamet) og de (de andre i teamet) har brukt en del par programmering. Utvikler-3 (en av utviklerne i teamet) begynte nå i oktober og utvikler-2 (en av utviklerne i teamet) har ikke drevet spesielt mye med dette. For å sette de i sving (og lære dem raskt opp), så bruker vi par programmering for å et kunnskapsløft. Et problem med dette er jo at det går litt tregere fremover også. Under sist sprint fant jeg og utvikler-1 ut at vi heller skal hive dem mer ut på dypt vann, ved å gi dem litt oppgaver som de skal gjøre for seg selv. Dette kommer de nok til å klare helt fint. Så kan de heller komme å spør (om hjelp dersom de trenger det). Men vi har brukt en del par programmering hvertfall.

Er det noen andre XP-teknikker dere har tatt i bruk?

Ikke som jeg kommer på. Ikke i teamet egentlig. Dette vet nok sikkert *scrum masteren* mer om enn meg. Det kan jo tenkes at det er andre team som bruker flere teknikker. S*crum master* vet nok dette. Han har mer kontrollen på det. *Scrum master* og release manager.

- *Nå har vi snakket litt om blant annet ulike sprint-lengder, er det noen andre ting som dere har endret ved måten dere kjører Scrum i løpet av den tiden du har jobbet her? Noen endringer for å tilpasse Scrum..?*

Det er jo dette med antall folk i teamene og sprint-lengde. Vi har skiftet fra antall dager til story points (når vi estimerer under planleggingen). Det er også en story point faktor som sier at 1 story point tilsvarer ca. 0.75 dager. Man planlegger dermed alt i story points istedet for i dager. Denne faktoren vil da bli bedre etterhvert. Det er hvertfall det som er planen. Det er *scrum master*, selvfølgelig, som er pådriver for dette her. Vi har også endret på en annen ting. Vi er blitt mye flinkere til å bruke tavlen. Jeg syns det er veldig viktig at man har en oversiktlig tavle. Tidligere hadde vi en backlog som kun var en plan i et Excel-ark og det var ikke helt optimalt. Disse to tingene er hvertfall blitt endret.

For eksempel sprint burndown chart illustrert ved en graf, var dette noe dere også brukt i begynnelsen, da dere startet med Scrum?

Vi hadde et burndown chart, men det var ikke så tydelig før. Da hadde vi det aldri på tavlen. Da kunne man beregne det utfra regnearket, men jeg kikket jo aldri på det. Det faktum at man har det på tavlen gjør at man hele tiden blir påminnet om hvordan man ligger an, man får en større tilhørighet til det (arbeidet i sprinten). Man blir ivrigere til å bli ferdige.

- *Når det gjelder review- og retrospektiv-møtene, hvordan er disse lagt opp? Hvem er det som deltar på disse?*

Det er *scrum master* og teamet som deltar på retrospektiv-møtene.

Product owner, for eksempel, pleier ikke å delta?

Nei, han (*product owner*) trenger ikke å være med da. Jeg tror han har mulighet til å være der, men *product owner* er jo veldig travel.

Er det noen andre som deltar av og til, for eksempel representanter fra kunden?

Jeg vet ikke om release manageren kanskje har vært med av og til. Det er som regel bare teamet og *scrum master*. Dette bør du spørre *scrum master* om. Spør han hvordan har pleier å kjøre disse møtene. Jeg klarer ikke å ta igjen helt nøyaktig alle skrittene, for jeg har ikke deltatt på disse møtene på over et par måneder.

Når du har deltatt tidligere, føler du da i etterkant at du får mye ut av disse møtene?

Retrospektiv-møtene er jo de kjedeligste møtene å gå på. Likevel på man jo ha dem for det er jo viktig å vite hva som er bra og hva som er dårlig. Selv om man kanskje sitter der og tenker; ”hva er dette for noe tull?”, så hender det at det kommer frem noe smart, for eksempel noe man må passe på. Det er jo under disse møtene man har mulighet til å kunne gjøre endringer (på hvordan sprinten skal kjøres).

Føler du at det blir gjort de nødvendige endringene eller tilpasningene i etterkant av disse møtene?

Vi prøver jo å følge opp de endringene vi har (kommet frem til), så etter neste sprint vil man jo se om problemene er de samme eller man har klart å forbedre seg. Vi får jo litt ut av de (retrospektiv-møtene), men som regel er de veldig kjedelige.

Når det gjelder demo-delen, når kjører dere den, er det en del av review-møtet?

Ja, først har du demo så har du retrospektiv-møtet. Under demoen så deltar jo *product owner* og da finnes det gode muligheter for å komme å kikke på (for personer fra andre team). I det siste har det også vært en del styr i forbindelse med at *scrum master* skal ha med user stories og tilhørende akseptansekriterier og den slags. Dette gjøres for at vi skal bli mer klare på "how to demo?" altså hvordan demonstrerer man dette. Hva er kravene for at det skal bli godkjent (av *product owner*)? Mer spesifikke krav må innhentes fra *product owner* fra begynnelsen.

Når dere holder på å skille ut oppgaver i forbindelse med sprint-planlegging, hvem er det som er mest aktive? Samarbeider dere en del med product owneren?

Det er *product owner* som setter sammen produktbackloggen, men nå i det siste har vi hatt pre-planleggingsmøter på forhånd for å kunne få frem bedre estimater over oppgavene og hva som faktisk skal gjøres. *Product owner* har jo ikke full oversikt. Han (*product owner*) vet hva som må gjøres, men det er viktig å vite hva som faktisk må gjøres for å kunne klare å fullføre det (oppgaven eller user stories) også. Vi studerer det litt og pre-planlegger gjennom diskusjoner med *product owner* på forhånd. Dette bør du spør de andre om. Det er jo de som faktisk har vært med på det. De har jo begynt med user stories i forbindelse med dette. Utvikler-1, utvikler-2 og utvikler-3 (utviklere på teamet) kan dette. De legger til akseptanse-kriterier og den typen ting på forhånd. Utfra dette kjører du sprint-planleggingen. Der har du en prioritert liste med oppgaver, så begynner du på toppen og så tar du en oppgave og deler den opp i småbiter og så estimerer vi hver av disse bitene og så summeres det opp hvor mye tid som vi kommer til å bruke på dette her (denne oppgaven). Så tar vi neste (oppgave) og fortsetter til vi ikke har mer tid igjen. Alle (på teamet) gjør dette sammen. Da kan vi også spille planning poker hvis vi vil det. Vi har lapper som vi pleier å bruke til det.

Pleier dere å spille planning poker?

Vi gjort det en del ganger. Hvis det er noe (en oppgave som skal estimeres) som vi er veldig enige om, så gidder vi ikke å spille planning poker. Dersom vi ikke er enige, så spiller vi planning poker. Da bruker vi (spesielle) kortstokker. Her har nok utvikler-1 (en annen utvikler) og de andre mye mer info, siden de har brukt det mens jeg har vært mer utenfor sprinten de gangene vi har har hatt med user stories. Det høres jo gjerne ut som mye dill for de som er på utsiden (folk som ikke driver med dette), men det ser faktisk ut som det kommer noe ut av det (bruk av user stories og planning poker). Det kan bli spennende å se. *Scrum masteren* er en skikkelig teoretiker på dette, mens *product owneren* har mer tankegangen; ”få det gjort.” Vi har pratet en del om kravene rundt hva man skal gjøre i det siste. Jeg tror det er lurt å ha noen klare kriterier for hva man faktisk skal levere. Det kan være lurt å få tenkt litt gjennom ting før man faktisk begynner å jobbe.

Hvordan syns du scrum master og dens rolle fungerer i forhold til teamet, som du er en del av?

Jeg syns det fungerer ganske greit slik det er nå. Det er alltid en kamp mellom *scrum master* og oss. For eksempel, skal vi følge boken eller ta snarveier. Noen ganger må man ta snarveiene for å bli ferdige eller fordi det kommer inn noe support.Vi klarer jo ikke å følge boken hele tiden, men det er jo en kamp med *scrum master*; ”jaja, vi skal gjøre det og det, men så blir det gjerne slik som vi alltid har gjort det.” Så vi har jo den evige kampen der, men det fungerer ganske bra. *Scrum masteren* er flink.

Er det slik at scrum masteren coacher dere eller delegerer han mer hvem i teamet som bør gjøre hva?

Vi velger alt selv. Hvem som skal gjøre hva styrer vi helt selv. Det han (*scrum master*) gjør er bare å være opptatt at vi gjør det vi skal. Han leder daily Scrum (daily stand-up møtet) og sørger for at alt går skikkelig for seg, at vi diskuterer det vi skal og hvis det er noen impediments, så tar han det opp med dem (det gjelder) for å prøve å fikse de (hindringene). Han (*scrum master*) kan ikke så mye om akkurat det vi holder på med, men han passer på at vi gjør det vi må gjøre. Dersom vi ligger langt bak, så er han bekymret og kommer og maser. *Scrum master* har også en forkjærlighet for testing, så han er en pådriver for det.

- *Er det slik at dere som regel skriver tester før dere begynner selve kodingen? Hvordan fungerer dette hos dere?*

I teorien så skal vi gjøre det. Vi skriver en del tester før vi koder, men tror ikke vi alltid gjør det. Spesielt for meg som driver med support så er det fryktelig fort gjort å fikse problemer fordi det haster og så tar jeg gjerne testene etterpå, men i teorien så skal man vel gjerne skrive, hvertfall en del, tester på forhånd. Det er ikke alltid det blir gjort. Noen ganger blir det gjort, mens andre ganger ikke.

Det har vært en del snakk om at test coverage skal være så og så høy hos dere, hvordan er det nå?

Det er det *scrum master* som har begynt med. På nye moduler så skal test coverage være så og så høy. Jeg vet ikke helt hvordan vi ligger an der, men jeg tror vi har klart å holde oss ganske greit i forhold til det. I de gamle modulene som ikke hadde (spesielt med tester), der ligger vi langt bak, men slik må det bare bli. Vi har ikke ressurser til å sette å skrive masse tester for gammel kode. Det kan vi bare glemme.

Vi skal egentlig følge test-drevet utvikling. Det er en del team som ikke har vært så flinke på for eksempel unit-tester fordi de gjerne ikke jobber med Java-kode og da blir det vanskeligere med en gang man skal skrive unit-tester. Utvikler-1 (en av de andre utviklerne) og *scrum master* er veldig ivrige på testing. Det er helst de som er pådriverne.

- *Når det gjelder dokumentasjon, har dere noen spesielle retningslinjer angående hvor mye eller når dere skal skrive dokumentasjon?*

Vi skal skrive dokumentasjon. Når det gjelder EC-IS så er skal vi skrive teknisk dokumentasjon, men det blir ofte for teknisk slik at når kunden skal bruke det så forstår de det ikke. Den ene oppgaven som de har nå i denne sprinten, er å skrive bedre dokumentasjon for EC-IS. Da skal det være mer brukerrettet; ”Hvordan bruker du dette?” i stedet for ”Hvordan virker det?” Når det gjelder EC-IS har vi fått mye pepper for dokumentasjonen fra kundene. Dette har nok generert en del support, bare dokumentasjonen i seg selv.

- *Er det noe du synes fungerer spesielt bra med slik dere kjører Scrum? Er det noe du har lyst til å trekke frem?*

Det er et veldig åpent spørsmål. Fordelen med å kjøre Scrum i forhold til en del andre ting er det at man er mye mer fri til å forandre. Da jeg jobbet i Trondheim for et annet selskap så planla vi prosjekter på en 6-måneds basis. Da hadde man en lang planleggingsfase først og så var den spikret. Så startet man å implementere. Nå er det mer slik at vi planlegger for 2-3 uker, så får man en ny backlog. Det er jo *product owner* som må holde orden på hva vi må gjøre og så bare plukker vi fra den. Vi blir mye mer dynamiske og kan endre mye raskere. Dersom et prosjekt trenger en spesiell endring så kan man gå inn å gjøre den med en gang, i stedet for at man må stoppe et langt løp for å re-planlegge. Det at vi er åpne for endringer er den store fordelen med Scrum.

- *Føler du at dere er mer smidige nå og klarer å tilpasse mer nå enn da du startet for to år siden?*

Vi blir flinkere til å bruke Scrum, det blir vi. Vi lærer mye underveis. Vi gjør jo en del endringer. Vi har endret team-størrelse, sprint-lengde og måten vi gjør en del ting på. Så vi blir jo stadig flinkere. Vi er bedre nå, men det er mye forbedringspotensiale enda.

Er det noen spesielle deler av hvordan dere kjører Scrum nå som du føler at dere kunne blitt bedre på?

Vi kan blir bedre på å ha krav (til oppgaver på produktbacklogen) klare på forhånd (før sprinten starter). Det er det vi prøver å få til gjennom user stories og det tror jeg at vi kan bli bedre på. Dette (vanskeligheter med å få klar krav på forhånd) tror jeg også henger litt sammen med at *product owner* har liten tid. Det skal godt gjøres å få et ekstra ledig møte med han.

B.2 Interview with *scrum master*/release manager in project-11, 23.February 09

❖ *Hvor mange er dere i hvert team?*

Hos oss har det nok variert over tid, men typisk team-størrelse hos oss nå ligger på rundt 5-7 personer. Vi prøver å holde oss til det som er anbefalt som er 7±2 og vi klarer dette på en del av teamene, men ikke på alle. Det er det som er målsetningen vår. Det største teamet har ni personer. Vi har faktisk to team som er ni personer, som sitter i Malaysia. Her i Stavanger så er vi f.eks fem personer på Framework-teamet eller så har vi et team som er fire personer, men målet er som sagt 7±2 (personer). Det er ikke alltid vi klarer det. Vi har ikke større team enn ni. Det kommer litt an på, men jeg heller ha et team på fem personer enn et team på ni personer. Store team er ikke alltid like lette å få til (å fungere).

❖ *Hvor lenge har hvert team jobbet sammen? Hva er typisk her og hvilke fordeler/ ulemper er det i forbindelse med utskiftning av team medlemmer?*

Vi har holdt på med Scrum i ca. 2.5år, og da vi startet med Scrum så hadde vi lenge i den første perioden ganske stabile team. Vi hadde de samme folkene på de samme teamene. Etter det har vi vært gjennom en periode, i fjor spesielt, fra sommeren og frem til november der det var veldig mye utskiftning. Det henger litt sammen med at vi har såkalte kunde-commitments, så en del av de ressursene, spesielt her i Stavanger, som hører til produktutviklingen er jo også litt innvolvert i prosjektene direkte ut mot kundene våre. Det har i grunnen vært kompliserende å holde teamene stabile. Det var også blant tilbakemeldingene vi fikk etter medarbeidersamtaler; at det var altfor ustabile team og at størrelse på teamene var for små. Da var vi enda færre personer på hvert team enn det vi er nå. Det er en målsetning å ha teamene stabile over flere måneder, kanskje over.. En ting som vi har endret på nå er at ett team kan jobbe på flere produktområder. Tidligere hadde vi et team på hvert produktområde, som egentlig gjorde oss ganske sårbare. Dersom et produktområde plutselig hadde veldig mye å gjøre, så fantes det ikke så stor fleksibilitet til å kunne holde teamene stabile for da shuflet vi typisk rundt på 1-2 personer. Det vi sier nå er at vi prøver å holde teamene stabile over tid og så sier vi også at teamene skal kunne jobbe på forskjellige produktområder, noe som betyr at de vil ha flere product managere som server dem (tjener teamene), men kun én *product owner*. Hos oss er product managere de som har ansvar for de forskjellige produktområdene og ofte har det vært én-til-én mellom product manager og p*roduct owner*. Nå har vi hvertfall ett eksempel, og vi

kommer til å få flere eksempler senere også, hvor ett team jobber både mot EC-DM (EC er produktet organisasjonen utvikler og vedlikeholder, DM står for Data Management) og EC-Framework. De har egentlig to product managere, hvor den ene av dem fungerer som *product owner* slik at teamet kun har ett punkt (å forholde seg til) når det gjelder avklaringer av prioteringer spesielt. Når det gjelder tekniske avklaringer så er det rettet mot product manageren. Dette gir en større sjanse til å kunne holde teamene mer stabile.

Kan du si litt mer om rollen til project manager (her skulle jeg egentlig si product manager, men sa feil) når det gjelder for eksempel hvordan denne skiller seg fra scrum master og product owner? Hva er de store forskjellene?

Vi har jo ingen project manager her hos oss. Jeg er vel den som er nærmest en slik rolle fordi akkurat nå har jeg (ansvar for) ett team, men så er jeg også release manager som egentlig går ut på å koordinere ting i forhold til releasene våre. Vi har jo releaser minst én gang i måneden. Vi har service packer som skal ut, pluss at vi har de store releasene. Det er mitt ansvar å koordinere at vi får levert i forhold til de kunde-commitmentene vi har. Men ellers så har vi jo ikke prosjekleder-rollen her. Det er vel ”release manager”-rollen som ligger testtest opp til den, men jeg krever jo ikke statusrapporter fra noen for eksempel, det er jo Scrum’ene. De skal jo være selvgående sammen med teamene vi har.

Én ting er jo dette med risikohåndtering som typisk faller på en prosjektleder, men vi har jo for eksempel risikohåndtering her også. Alle disse områdene som en prosjektleder normalt gjør, der risikohåndtering er én av dem. Teamene håndterer det som potensielt kan være en risiko. Impedimentene er det jeg anser som risiko i teamene, som teamene sammen med *scrum master* normalt løser. Men så har du for eksempel risikofaktorer som kommer utenfor teamene, her må jeg finne et godt eksempel på en risikofaktor. Uansett om det er risikofaktorer som på en måte må skaleres, så er det typisk mitt bord dette havner på. Da diskuterer jeg det med en av product managerne som også er sjef for hele utviklingen og så tar vi det opp i styringskomitéen, som vi har månedlige møter med. Det er jo også en sak (styringsmøtet) som ligger utenfor Scrum. Selv om Scrum passer veldig godt til å drive team og arbeid fremover, så må man på en måte ha en helhet på det i forhold til kontraktene vi har og at vi har alle stakeholderne som kan være med på å ta bestlutninger. Dette er oppå Scrum, på en måte, slik at vi styrer frem mot noe. Sånn sett så kan du jo si at har et Governance, en styring rundt det vi gjør på release-arbeidet. Vi har jo et overbygg på Scrum.

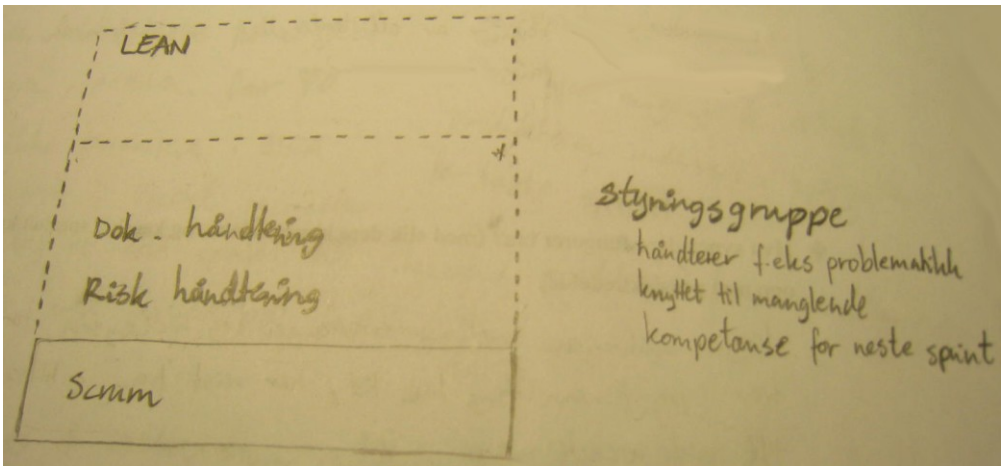

Figure B.1: Styringskomitéen/styringsgruppen i prosjektet.

Vi har vår Scrum med de forskjellige teamene her. Så har vi jo et overbygg som egentlig går på risikohåndtering, kontrakthåndtering og dokumenthåndtering. Det ligger jo litt på utsiden av teamet selv om teamet benytter seg av det vi tilbyr. I tillegg så er det en del tradisjonelle prosjektlederoppgaver som vi håndterer hele tiden. Du kan si at de risikoene som ikke håndteres her nede (peker på Scrum-blokken), de blir da håndtert her oppe (peker på styringsgruppe-blokken) i den styringsgruppen vi har. Der diskuterer vi dem og tar aksjoner på dem. Det kan typisk være et team som sier at de mangler en spesiell kompetanse for det som kommer i neste sprint eller de klarer ikke å levere fordi de ikke har den kompetansen de trenger i teamet, de trenger mer ressurser. Det kan ikke teamet løse selv. Det må vi eskalere og det er en typisk sak som gjerne ikke er et godt eksempel på en risikofaktor, men det er klart risikoen ved å ikke gjøre noe her er jo at du får et demotivert team og du klarer ikke å levere noen ting.

Du skal jo ikke skrive deg ihjel. Det er jo litt av idéen med at du ikke skal ha unødvendig dokumentasjon, men det er klart at noen slike formelle ting er vi pålagt av organisasjonen også. Så når vi kjører prosjektene så skal vi følge disse her. Dette må vi etterfølge for vi får revisjoner her. Vi prøver å holde det (dokumentasjonen) på et minimum. Likevel så er det noe som du må ha håndtert på en eller annen måte. Du må hvertfall ta stilling til det. Så har du dette med lean som er mer en filosofi som vi har sett på, men vi har ikke gjort noe med det. Det går jo mer på å effektivisere produktkjedene din og gjør en del analyser på hvor flaskehalsene er. Det gjennomsyrer hele organisasjonen og det er typisk en top-down sak, der ledelsen må, på en måte, kutte alt som har med waste å gjøre. Det kan være å kutte alt

som har med unødvendig dokumentasjon, eller for eksempel påbegynte oppgaver som ikke blir fullført. Vi er ikke kommet dit enda for å si det slik.

- Hva ser du på som din viktigste oppgave som *scrum master* (utfra rollen din under en Scrum)?

Hjelpe teamet til å nå sine mål.

Er det noen spesielle tiltak du gjør for å oppnå dette?

Det er jo forskjellige approacher (tilnærminger), men det kommer an på hvilken type team det gjelder. Dersom det er et team (hvor team medlemmene er ganske) erfarne, altså som har jobbet en stund sammen, som er i stand til å gjør det aller meste uten å bli fasilitert veldig mye.. Dersom jeg har et slikt team så vil jeg normalt la de få jobbe så mye som mulig i fred og egentlig bare sørge for å fjerne eventuelle impediments, men ellers bare la dem få jobbe i fred. Nå har jo ikke jeg hatt særlig mange ferske team, altså team som mer eller mindre starter fra scratch. Der vil vi jo gjerne ha en litt annen tilnærming. Der vil vi gjerne utfordre dem mer. Det går jo også gjerne en del på personlighet; Hvilken type man er som *scrum master*. Hvilken stil har du? Det finnes ingen fasit her. Det viktigste er hvertfall at du er der til å hjelpe teamet med å nå sine mål. Så er det jo viktig hva slags ting du skal bruke i forhold til hva slags team du har med gjøre og hvilken type oppgaver teamet har. Det vil jo også variere. Noen team har ganske fast definerte ting som er enkle å forholde seg til for teamet, mens andre igjen jobber på upløyd mark og da trenger man en annen approach for å få fremdrift på et slik team enn i et team hvor alt er forhåndsdefinert.

- Vi var inne på dette litt tidligere, men vi kan jo se litt mer på hvordan deres bruk av Scrum har endret seg etterhvert som tiden har gått, i løpet av de 2,5 årene dere har brukt Scrum. Jeg ser for meg at etterhvert som dere er blitt mer komfortable med Scrum så har dere gjerne tonet ned eller kuttet ut noen teknikker og gjerne lagt til noen andre, kanskje spesielt i og med at dere i nyere tid har begynt å fokusere mer på blant annet user stories. Kan du si litt om det?

Du kan si at da vi startet med Scrum så hoppet vi (ut) i noe ukjent. Det kan vi godt si. Vi fikk det til å fungere rimelig greit mer eller mindre med en gang, noe som gjorde at vi valgte å satse på det (Scrum). Dersom vi ser tilbake noen år, så er det klart at vi gjorde en del klassiske feil, tror jeg. En av de feilene vi gjorde var nok at vi ikke tok opplæring

alvorlig nok. På grunn av at prinsippene er såpass enkle når man leser dem at det (gir inntrykk av) at det ikke krever så veldig mye opplæring. Vi hadde opplæring av en del *scrum mastere*, men det vi glemte var opplæring av *product ownerne* og at folk i organisasjonen må forstå prinsippene. Det ser vi jo egentlig nå at vi ikke gjorde den gangen. Hvertfall det siste halvåret så har vi hatt flere kurs her. Vi har hatt kurs for alle *product ownerne*. Vi har hatt flere *scrum mastere* på kurs, pluss at han som er leder i Malaysia pluss to av *scrum masterne* her på kurs nå i forrige uke. Da de kom tilbake så hadde de sett ett helt annet lys. Det er vel én av de store tingene som vi gjorde litt for sent. Det var én av de viktige tingene som vi ikke tok alvorlig nok da vi gikk i gang.

En annen ting vi gjorde var at vi hadde weekly (ukentlig) Scrum of Scrums. Det har vi kuttet ut nå, fordi vi har ikke fått det til å fungere bra, men vi har heller ikke noen god erstatter for det. Der må vi se på hvilke måter vi kan...

Hva var grunnen til at dere følte at Scrum of Scrums ikke fungerte?

Jeg tror at det går litt tilbake gjen til forståelse av hva Scrum er. Når du skal ha en Scrum of Scrums så vil du jo gi et bilde av hva det er teamene holder på med, på et visst nivå. Du er ikke interessert i å gå ned på et issue-nivå der du sier at den personen holder på med det osv. Du vil gi et bilde av hva slags funksjonalitet er det de (folk) jobber med, hvilke problemer har de osv. Altså på dette nivået, ikke på et lavt nivå. Det har vi bare rett og slett ikke fått til å fungere. Vi satt gjerne 6 personer sammen i en halv time og brukte unødvendig tid. Der må vi få et alternativ på plass. Nå har vi ikke fått den kommunikasjonen til å skje godt nok, vil jeg si. Det har ikke gitt de som har vært med på disse møtene nok til at det er forsvarlig å kjøre dem. Det vi har gått i gang med istedenfor, som et supplement.. Det er jo klart at det vil jo ikke erstatte det som er hensikten med Scrum of Scrums, men vi har hvertfall et møte annen hver uke med alle *scrum masterne* – et koordineringsmøte. Dette møtet går på å align'e *scrum masterne* med hva som skjer i de forskjellige teamene og typiske problemstillinger som en *scrum master* har. Typisk også måter vi gjør Scrum på, forbedringer vi må gjøre eller hvis det er problemer eller hva det måtte være som gjelder *scrum masterne*, så tar vi det opp her.

Jeg har inntrykk av at dere er flinke til å sørge for at dere holder dere til Scrum-prinsippene og da jeg var inne på et av team-rommene, la jeg merke til en forholdvis ny og oppdatert bok om user stories som lå der. Har dere noen spesielle rutiner for å holde dere oppdaterte eller for å passe på at dere fortsatt følger Scrum, siden det fungerer såpass bra?

Nei, det har vi ikke. De bøkene er det stort sett jeg som har bestilt enten basert på forespørseler i fra folk her eller fordi det er bøker som jeg ser kan være nyttige, som folk kan bla i. Vi har ikke noen spesielle rutiner. Det er klart at dersom vi ser på måten vi har kjørt Scrum på og kjører Scrum på, så ser hvertfall vi som holder på med dette til daglig alle userne vi står ovenfor og alle forbedringene vi skal gjøre. Men vi skal heller ikke glemme at vi i over 2,5 år har levert releaser ut til ganske krevende kunder og at vi faktisk hver måned, mer eller mindre, leverer ut service packs som også er en ganske stor release. Vi gjør mye riktig, men jeg tror at vi ikke tar ut hele det potensialet som ligger i Scrum. Se for deg at vi svitsjet (byttet) fra en tradisjonell vannfalls-måte, den var jo ikke helt vannfall heller for den var til dels iterativ, men vi gikk hvertfall over fra en mer traidjonell måte å utvikle på til en agil (smidig) måte å utvikle på, altså Scrum. Så vil jeg jo si at vi ble litt bedre, men ikke vannvittig mye bedre. Vi klarer å levere og vi klarer sannsynligvis å levere noe bedre enn det vi gjorde tidligere. En av utfordringene er at vi ikke har noen målinger som er veldig gode på; ”så gode er vi nå i forhold til tidligere.” Jeg tror at vi kan bli betraktelig mye bedre ved hjelp av en del grep som vi holder på å gjøre nå. Fokuset vårt er derfor å prøve å ta ut dette potensialet for å få det til å bli en bedre plass å være for de som jobber her.

Du nevnte i sted at dere i den siste tiden har hatt en del kursing innad i bedriften, hvem er det som vanligvis holder disse kursene eller hvordan gjør dere dette?

Ja, det er jo slike standardkurs, sertifiseringskurs for *scrum mastere* og sertifiseringskurs for *product ownere*. Det har vi kjørt, pluss at vi også har leid inn et selskap fra Sverige, som har vært her i to runder. Vi skal ha én runde til med dem. De er konsulenter i bransjen, kan du si, som tar på seg slike oppdrag til å kunne være med å redusere problemer, tuning av Scrum prosessen eller engineeringpraksiser som ligger i bunn.

Dere har jo nylig innført bruken av user stories og kan du fortelle litt mer om selve sprint-planleggingen? Ser du noen spesielle utfordringer når det gjelder estimering?

Det er nok litt ulike praksiser i de forskjellige teamene. Vi har de aller fleste team lokalisert her i Stavanger. Vi har jo team som er rene Stavanger-team, vi har team som er rene Malaysia-team og så har vi to team som er kombinerte med folk fra begge steder. Det varierer nok litt i fra team til team. Hvis ser litt på hvordan teamene her jobber, så vil en typisk si at en sprint-planlegging starter med issue-gjennomgang. Dette gjelder hvertfall for begge de halv-store teamene her. De går gjennom issuene før planleggingen for å få en forståelse av hva du skal gjøre, så vil teamet selv estimere enten det er i story points eller det er i timer, det er gjerne ikke så relevant her. Det er jo bare to måter å se det på. Det er den ene måten vi gjør det på i flere av teamene. Så har vi den andre måten, som gjelder spesielt det teamet som er et rent Malaysia-team, hvor *product owneren* sitter her. Da har *product owneren* en formening om hvor lang tid dette skal ta og så vil han da sette opp timer på hvor mye, sånn anslagsvis, han tror det vil ta. Så vil det være teamet som da går gjennom og tar inn hans estimater, men det er klart teamet vil i de fleste tilfeller gå inn og stille spørsmål i forhold til estimatene og sette sine egne estimater på jobben. Det er jo teamene selv som best vet eller bør vite hvor lang tid ting vil ta. De kjenner hvem de har i teamet og de kjenner forutsetningene til den enkelte osv. De vil gi fasiten, selv om et estimat er et estimat, så er det de som kan gi det beste estimatet. Det får vi se rimelig bra her i Stavanger, men det går kanskje også litt på forståelsen i Malaysia, at de gjerne ikke tenker på samme måte. Det var jo derfor at vi nå har hatt de tre på kurs nede i Malaysia, så nå har de kommet tilbake igjen. Vi hadde et møte med dem i dag der de var ganske begeistret for mange ting, men blant annet det med å få den forståelse, for det er jo teamet som er den beste instansen til å kunne si hvor lang tid ting vil ta. Der er det nok litt variende, alt etter hvor du er i forhold til estimering, men vi bruker både timer og story points. Vi har ingen slike; ”dette er måten å gjøre det på” uttalt (interne standarder). Det kan godt være at vi skal gjøre noe med det, men foreløpig er det slik.

Hender at dere for eksempel bruker planning poker eller slike teknikker i forbindelse med estimering?

Ja, det gjør det.

Er det spesielle tilfeller hvor dere bruker det, eller foregår dette hele tiden..? Er det noen spesielle fordeler ved for eksempel å bruke planning poker?

Fordelen er at du får en dialog med hele teamet. Tingen er jo at dersom teamet har forskjellige oppfatninger av hva som er jobben som skal gjøres, så vil du kunne identifisere dette og få alle involvert. Det er jo klart én av fordelene ved planning poker å få de som normalt gjerne ikke sier så mye til å si noe. Typisk hvis en hiver på en 2-er og en hiver på en 10'er (under planning poker), så blir jo spørsmålet hvorfor hev du på en 10-er og du en 2-er? Snakker dere om samme tingen? Så får du en dialog, ikke sant. Det er jo et veldig sterkt verktøy når du snakker om estimering. Nettopp derfor vil du få det beste i fra teamet ved for eksempel å bruke planning poker eller andre måter å gjøre det på. Det viktigste er jo at du får den dialogen rundt det som skal gjøres. Planning poker er jo bare en teknikk for å få det til.

- *Hvordan er review og retrospektiv-møtene lagt opp i dag? Hvordan fungerer de nå og hvem er det som deltar? Føler du at problemer som blir identifisert blir tatt tak i og endret til neste sprint slik at dere hele tiden er smidige eller føler du at man kanskje alltid får så mye ut av disee møtene?*

Jeg tror retrospektiver har flere hensikter. En hensikt er selvfølgelig å se på hva kan vi gjøre bedre i neste sprint. Det andre er egentlig bare å få ut det som du har gjort før du starter på noe nytt. Hvis du snakker om de forbedringene som en ser som følge av en retrospektiv, så vil det gjerne være en todeling. Det kan være for eksempel ting som teamet selv ser at de kan gjøre på en annen måte. Så har du også de tingene som ligger utenfor teamet, for eksempel at vi mangler en server. Denne trenger vi for å jobbe bedre. Vi så vel det for en tid tilbake, at vi ikke utfordret teamene nok. Da hadde vi jo 4-ukers sprinter. Nå har vi 2-ukers sprinter. Da hadde vi sagt på retrospektiv-møte at vi skal gjøre dette, og dette, og dette, så gikk det fire uker og så ble de ikke konfrontert eller de glemte det kanskje. Det er hvertfall én av de tingene vi skal ta opp på sprint-planleggingen: "dette var det dere sa på retrospektiven, som vi hadde for én dag eller to siden" og så blir det også annonsert i retrospektiven så kan du gå tilbake og se hvorvidt det skjer da, men det er hvertfall det vi skal gjøre i forhold til Scrum-rammeverket.

- *Vi kan kanskje snakke litt mer om det faktum at dere endret sprint-lengden. Grunner til det og hvor lenge dere kjørte 4-ukers sprinter?*

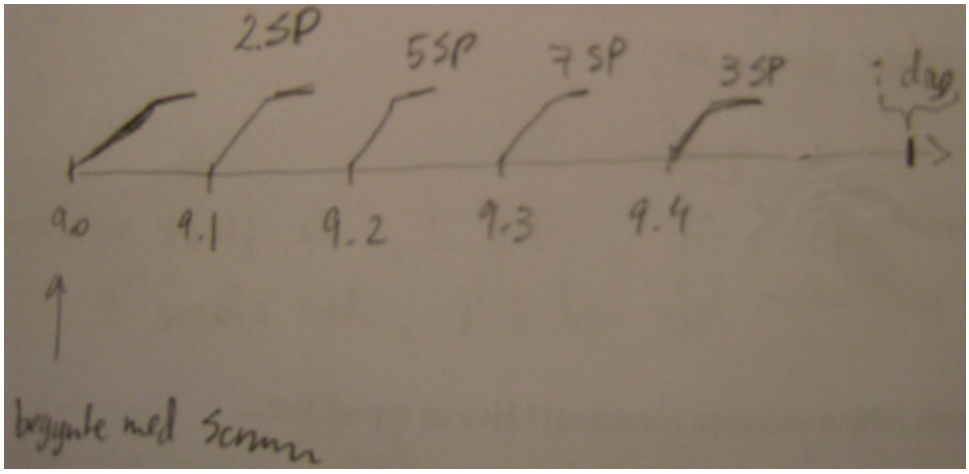

Figure B.2: Versjoner og service packer siden innføringen av Scrum.

Her er kodebasen vår. Her ligger all vår kode som er lastest and greatest. Her kommer release 10.0. Da vi startet med Scrum, så var vi på versjon 9.0. Så har vi i løpet av de 2,5 årene levert fem ganske store releaser. I tillegg så branch'er vi ut når vi leverer en release, da branch'er vi. Det gjør oss i stand til å kunne levere ut til flere kunder på for eksempel (versjon) 9.2. Ok, jeg trenger en patch. Da må vi kunne gjøre det på slik koden var i utgangspunktet i versjon 9.2 (i dette tilfellet). Vi kan ikke gjøre det på den koden vi har nå. Vi må gjøre det på den koden de kjører, slik at vi shippe en patch på det opplegget de har. Tingen er at på (versjon) 9.1 så har vi to service packer. En samling av patcher og en samling av ny funksjonalitet blir typisk en service pack, der du har ganske mange nye ting og en del bug-fikser. Her (peker) tror jeg vi har seks service packer, her (peker) kommer vi ut med nummer syv. Det betyr at da vi startet her (peker) så hadde vi en ganske enkel line. Da jobbet alle teamene på head, altså på main line, på det som skulle bli den neste releasen. Vi har ett team som jobber med maintenance (vedlikehold) dvs. jobber bare med fikser og gjør mindre forbedringer på gamle versjoner som ligger bak her (peker). Vi har flere også. Det er en ganske stor jobb, og det er jo det jeg gjør da; Å planlegge dette med service packene slik at de kommer ut i tide slik at kundene våre blir fornøyde, slik at de får med de tingene som de forventer at skal bli fikset og ny funksjonalitet. For å kunne koordinere dette her, så må vi gå ned på sprint-lengden. Da vi hadde 4-ukers sprinter så hadde vi normalt en kodefrys og så hadde vi demo og så ny sprint-planlegging og så (en) ny runde. Det var kodefrys osv. Det er vanskelig å time (beregne) dette her med leveransen

av disse service packene og for å få til det, så fant vi ut at vi måtte gå ned på sprint-lengden til to uker. Da vil vi være i mye bedre stand til å kunne planlegge inn mot de forskjellige service packene. Kodefrysen på en sprint er da linket opp mot den ”service pack”-leveransen. Det var en av hovedårsakene til at vi gikk fra fire til to uker (når det gjelder sprint-lengde). Det var den ene årsaken. Den andre årsaken var at vi har hatt problemer med å få en god planleggingshorisont. Det er noe som vi har en utfordring med nå i dag også. Dersom det står en eller annen *product owner* her; ”Hva skal du levere for ditt produktområde?”, ”Hva er det som kommer om to måneder?”, altså i den sprinten som starter om halvannen til to måneder. Da ville jeg gjerne hatt et svar som at; ”Slik det ser ut nå så vil det være disse nye featurene pluss noen bug-fikser som kommer med, men det kan endre seg selvfølgelig.” Men vi har ingen slike horisonter. Vi ser egentlig bare fortsatt 2-3 uker frem i tid. Det er et problem. Det henger sammen med at vi har ingen god velocity på alle team og vi har heller ikke gode nok produkt backloger på forbrukerne sin side som gjør det at vi kan se frem tid slik. Det var den andre årsaken. (altså; har kortere sprinter, 2 uker, for å lettere kunne samkjøre vanlige sprinter med service pack releaser, men også for å opparbeide pålitelige velocity-metrics for alle team og for å sikre at produktbacklogen blir oftere vedlikeholdt og dermed får høyere kvalitet, slik at planleggingshorisonten blir forbedret.)

I og med at dere har ganske korte sprinter så ser jeg for meg at det kan være viktig å ha en ganske kort sprint-planlegging slik at det ikke blir en for stor del av selve sprinten. Hvor lang tid tar det typisk å planlegge en sprint?

En tommelfingerregel er vel, i hvertfall i teorien, én time pr. sprint-uke. Jeg tror at vi bruker litt mer tid enn det. Slik at en sprint på to uker, si ti dager da, så bruker vi kanskje en halv dag, i effektiv tid. Det kan være at det er litt for mye, og det henger nok sammen med at.. Det er litt etter hva du ser på som sprint-planlegging også. Da må vi holde utenfor issue-gjennomgangen (pre-planning), som egentlig er en del av det som teamet skal være med å hjelpe *product owner* med, det å faktisk planlegge frem i tid. Nå tenker vi bare på hva vi skal gjøre i neste sprint. Vi er ikke kommet lenger enn det. Da går det egentlig på å detaljere. Vi har jo backloger som lister masse issues, men da er det gjerne masse detaljer som må være på plass før en er klar til å ta det inn i en sprint. Den issue-gjennomgangen er jo det som teamet gjør for å hjelpe *product owner* til å få en (produkt) backlog som er i shape. Det kan være noe som gjerne pågår i løpet av en sprint, med ting som kan komme i

neste sprint. Så har du selve sprint-planleggingen. Det varierer nok litt det kan være alt i fra to til fem timer alt etter hvilket team det er og hvor klart ting er. Det som er typisk dersom du kommer inn med en backlog som er lite gjennomarbeidet, så vil en slik sprint-planlegging fort kunne dra ut i tid. Da blir det masse avklaringer og du borer deg ned i diskusjoner som egentlig burde ha vært unnagjort før sprint-planleggingen. Men det går litt på det at våre *product ownere* er flaskehalser i dag. Det er egentlig to ting. Vi har ikke god nok verktøystøtte for det slik at de kan jobbe effektivt med produkt backlogen. Det andre er at de (*product ownerne*) har så mye annet (å gjøre) at det med å planlegge ofte blir utsatt til siste liten og da får du en produkt backlog til en sprint-planlegging som ikke er god nok.

Hvordan er det dere håndterer produkt backlogen nå? Er den for eksempel synlig for alle teamene som har interesse av den?

Nei. I utgangspunktet ikke. Den er jo synlig i den forstand at den ligger i vårt issue-håndteringssystem, som er Jira. Det er synlig for alle som har tilgang til det systemet, men problemet er at i det verktøyet, Jira, så er det ingen god måte å gjøre prioriteringer på. Du kan ikke si; ”Gi meg de fem viktigste issuene.” Det kan du ikke ta ut sånn uten videre. Det vi gjør i sprint-planleggingssesjonene våre er at vi eksporterer ut issues fra Jira til regneark og så sitter *product owner* og manuelt gjør en prioritering, og setter dem opp i den rekkefølgen de skal være i til sprint-planleggingen.

..Så sender product owner dem ut på for eksempel e-post da eller?

Så sender han (*product owner*) dem ut før sprint-planleggingen til teamet sitt. Problemet med det er jo at det er godt kjent mellom *product owner* og teamet hvilke prioriteringer vi snakker om. Derimot hvis jeg som release manager skal gå inn å se på alle top 10 issuene på alle våre produktområder, vi har altså 6-7 produktområder, så må jeg i beste fall inn i 6-7 forskjellige regneark å lete meg frem til det og det har jeg jo ikke tid til å gjøre. Det jobbes en del med verktøystøtte for å få et verktøy som kan prioritere og som henger sammen med Jira. Det er kanskje den viktigste tingen vi gjør akkurat nå, fordi det henger sammen med så mange andre ting. Det går på det at ting skal være transperente. Hvis for eksempel et av prosjektene våre kjører mot en eller annen kunde og de er i en aller annen testfase, ting skal gjerne settes i produksjon om ikke så lenge, så er det jeg som får forespørslene om f.eks; ”Denne her Jira-issuen, som jeg registrere for to uker siden eller for to måneder siden, kommer den med eller ikke?” Det har jo ikke jeg nubbesjangs til å

kunne svare på, på strak arm fordi det ligger i den store hopen. Det ligger ikke i en prioritert kø. Da begynner vi å snakke om de tingene som er veldig vanskelige å få til når det gjelder Scrum. Det at ting skal være transperente. Det vil si at internt i egen organisasjon skal ting være såpass åpent at alle kan få et innblikk i hva som skjer. Kundene våre på prosjektene, altså det som kjøres ut mot store oljeselskap (for eksempel), de skal også kunne gå inn å se på dette for å få en forståelse av de tingene som de venter på (da kan de se på); Hvor ligger de (forespørslene deres) i køen? Der er vi ikke og det er jo en fundamental ting som vi først nå klarer å gjøre noe med. Alle møtene vi har og alle Scrum'ene vi har er jo åpne. Folk kan bare ringe inn på de gitte tidspunktene å høre på møtene og komme inn på Scrum'ene osv, men det er bare en liten del av det som går på at ting skal være transperente.

- *Nå har vi snakket om forskjellige oppgaver som dere står ovenfor når dere jobber med Scrum og forskjellige prosjektledelsesoppgaver. Hva vil du si er de viktigste oppgavene dere står ovenfor i forbindelse med prosjektledelse (innenfor Scrum), hva er hovedutfordringene når det gjelder prosjektledelse (innenfor Scrum)?*

Mye av det som lå under prosjektledelse gikk jo på planlegging, statusrapportering og oppfølging. Det er noen av de tingene jeg har gjennomgått tidligere som prosjektleder. Det aller meste av dette er jo nå flyttet ned til teamene. Det er de som rapporterer og det er (i grunnen) status du får (vite) på daily Scrum (daily stand-up møter); "Hva har jeg gjort?", "Hva planlegger jeg å gjøre?" og "Hvilke problemer har jeg?" Det er jo en deling. Du kan si at *product ownerne* har også en rolle som prosjektlederen hadde, nemlig dette med å dele ut oppgaver, assigne oppgaver. Tingen er jo den at de (*product ownerne*) assigner ikke. Han (*product owner*) kommer med lister som teamet plukker i fra (frihet under ansvar eller innfor visse rammer). Så det er ingen som kommer å sier; "Dette forventer jeg." Det bør jo ikke være slik hvertfall. "Dette forventer jeg levert i løpet av to uker" og så kommer du med en lang liste som teamet ikke har sjangs til å klare. Før var det gjerne mer slik at man kunne gjøre det med bedre samvittighet, kanskje. Men nå, dersom du har gode team så får man en stopp på det. Dersom du prøver deg med det, så vil ikke teamet godta det. For det er jo de (teamet) som sier; "Ja, vi skal levere." Det må være realistisk i deres øyne at de skal klare det.

Det vil jo være lettere å se om det er realistisk i og med at du har kortere sprinter og det dermed er kortere tid til du faktisk skal levere. Da tar det mindre tid å finne ut om den listen du får levert og de oppgavene du skal gjøre faktisk er mulig å ferdigstille innen den gitte tidsfristen enn før Scrum, da du gjerne måtte vente 6 måneder. Da ville det være vanskelig å si; "klarer vi å gjøre denne listen med oppgaver innen 6 måneder eller ikke."

Helt klart. Det er jo ikke tvil om at akkurat det er én av styrkene til dette (til Scrum). Det med å bruke mye tid på å planlegge ting som ligger et godt stykke frem i tid og hvertfall detalj-planlegge ting som ligger tre, fire, fem, seks måneder frem i tid er waste of time (bortkastet tid), fordi det skjer såpass mye før den tid kommer. Det betyr at du planlegger hyppigere og du planlegger kun for de neste to ukene, hvor du er rimelig sikker på at du treffer. Planlegging og oppfølging var noen av tingene som en prosjektleder typisk gjorde, det er det nå for det meste teamet som gjør. Det er jo litt vanskelig å si. Prosjektlederrollen er jo nå fordelt utover tre forskjellige roller. Det er *scrum master*, *product owner* og team.

- *Når det gjelder dokumentasjon, har dere noen spesielle retningerslinjer når det gjelder mengde eller tidspunkter dere skal jobbe med å dokumentere?*

Ja, det har vi, men det er ikke alltid vi følger det. Men retningslinjene er at når vi lager nye skjermbilder så skal det lages et dokument som beskriver både funksjonelt og detaljert et design hvor deler av dette dokumentet vil inngå i dokumentasjonen som vi leverer ut med releasen. Det er jo egentlig *product ownerne* som tar den avgjørelsen angående det å lage ny dokumentasjon for ny funksjonalitet. Det ligger i bunn som en regel som nok sikkert vil kunne fravikes.

- *Er det noen spesielle XP-teknikker som dere benytter dere av som for eksempel par programmering? Er det noen teknikker dere bruker oftere enn andre eller kun i spesielle sammenhenger? Vil dette variere fra sprint til sprint, eller fra prosjekt til prosjekt?*

Vi har jo sagt det at når det gjelder par programmering eller code reviews, vi kaller det peer review altså review av en likemann, så skal det gjøres på all kode. I det legger vi at det skal enten ha vært gjort en code review eller så skal kodingen foregå ved pair programming. Det ligger definert i vårt Scrum-rammeverk at slik skal det være. Det er den ene tingen vi bruker fra XP. Vi bruker egentlig ikke test-drevet development, vi har kanskje brukt det til dels, men det er ikke noe som gjennomsyrer måten vi jobber på

egentlig. Det er et av de forbedringsområdene vi har. Vi bruker contiuous integration, men der må vi få gjort mer. Vi har automatiserte tester og bruker CruiseControl. Vi har nattlige bygg på head eller mainline og så har vi det på (versjon) 9.4. Vi har jo skrevet mange tester, men vi har en veldig lang vei å gå der. Sånn er det jo typisk når du kommer i gang såpass sent. Vi har godt over 1 million kodelinjer og når du begyner med automatiserte tester vil det alltid være mye å gjøre i forhold til det som du klarer å få med deg. Det er jo klart at det er en utfordring i forhold til hvis vi gjør noe et eller annet sted i vårt produkt, så er det i enkelte tilfeller vanskelig å vite hvilket impact det har på andre ting (andre deler av produktet). Da er vi sårbare hvis vi ikke har automatiserte tester som plukker opp feil og kan se at ting fungerer selv om jeg ikke rørte den delen av koden.

Det (test-drevet utviling) har nok skjedd til en viss grad i hvertfall i ett av teamene som vi har hatt her, men det er ikke noe som er innarbeidet. Det er en stor sak som tar tid å innføre og vi har ikke satt nok fokus på det egentlig.

Jeg kan se for meg at på grunn av at folk har forskjellige programmeringsvaner, så vil det å snu om til en test-drevet utvikling, på mange måter, være å snu om på kodekulturen i bedriften og det er jo klart at det vil være vanskelig.

Vi gjorde jo et valg når det gjaldt arkitektur for lenge siden og det er jo ikke sikkert at det er alt som passer like godt heller i forhold til å jobbe på den måten (test-drevet utvikling) der.

- *Jeg tror i grunnen vi har vært innom det meste, men vi kan jo gjerne helt til slutt se på om det er noe du vil trekke frem som du synes dere gjør veldig bra med slik dere praktiserer Scrum nå, noe dere er veldig flinke på.*

Vi klarer å levere software som stort sett fungerer bra og har gjort det over såpass lang tid, så noe har vi fått til. En av de tingene som har vært bra, men som for all del alltid kan bli bedre, er kommunikasjonen mellom Stavanger-kontoret og Malaysia-kontoret. Det er ikke tvil om at vi har fått et vannvittig stort løft der. De er blitt mye flinkere på å kommunisere. Det at det ikke er noen ”vegger” mellom oss, at de tør å spør oss. De sier i fra dersom ting tar tid. Tidligere så kunne de gjerne ha en forespørsel som lå der nede (i Malaysia) i dagsvis, gjerne i ukesvis. Nå er det ganske synlig hvis det mangler avklaringer og da er det en annen forståelse for hva konsekvensene er ved å la ting ligge å vente på avklaringer. Det er jo en veldig bra ting.

Hvordan er det denne kommunikasjonen foregår? Er det via telefon, videokonferanse, e-post eller instant messaging?

Det er en kombinasjon av flere av disse tingene. Det er klart at vi over tid har hatt folk her oppe mye (folk på besøk fra Malaysia), det har vært folk her i fra nede (på besøk i Malyaisa) pluss at mye går på communicator (Office Communicator), instant messaging og morgenmøtene (daily stand-up møtene) gjøres via telefon med webcamera. Vi bruker nettmøter når vi har sprint-planlegging og demo med sånne "discware"-ting.

B.3 Interview with *scrum master* in project-11, 23.February 09

- *Hvilke utfordringer ser du i forbindelse med prosjektledelse og din rolle innenfor Scrum?*

Den største utfordringen i forhold til rammeverket (Scrum) er at ressursene kan bli omprioritert underveis. Så man kan klage på at kanskje Scrum ikke er forankret helt til topps i organisasjonen. Det er de største utfordringene mot metoden.

En av scrum masterne deres nevnte at dere har noen møter som involverer forskjellige scrum mastere og høyere ledelse, hvordan fungerer dette?

Det er et styrende møte der vi som *scrum mastere* diskuterer de problemene vi har. Først status; Hva har vi gjort siden sist møte? Så diskuterer vi litt hva som bør gjøres for å bli flinkere. Vi diskuterer måter og hvordan løse de problemene (som blir tatt opp). For eksempel: "Er det et pattern vi kan bruke for å skalere det her?" Da kan man også diskutere om vi skal ofre én/sacrifice one. (La oss si at) Vi har et problem og det er i ferd med å ødelegge hele Scrum'en og da ofrer vi kanskje én og tar det problemet ut av Scrum'en. Vi diskuterer måter å løse praktiske eller daglidagse/vanlige problemer på. Det er et fora for *scrum masterne* for å diskutere problemer. Hvordan skal vi løse disse (dele) erfaringer.

En av scrum masterne snakket om at tidligere så hadde dere forsøkt å kjøre ukentlige Scrum of Scrums, men dette hadde ikke fungert helt.. stemmer det?

Idéen var at én fra teamet skulle møte opp, én fra hvert team. Det var ikke samme (person) hver gang og det sier seg selv når det sitter sju team og hvis det var seks i hvert team, så var det aldri de samme som traff hverandre. Nå er dette møtet på "*scrum master*"-nivå, og

jeg syns det er mye bedre å gjøre det sånn. Vi har mistet den biten vedrørende det at hvert team skulle vite hva de andre teamene holdt på med.

- *Da lurer jeg på sprint retrospektiv-møtene deres. Hender det noen gang at det er folk fra andre team som er med på disse møtene?*

Det har skjedd, det har hendt bare én gang. Men det er litt oftere nå enn for 7-8måneder med tanke på at *product ownerne* deltar på retrospektiv-møtene. Vi har også blitt flinkere på å faktisk følge opp de tingene (som blir tatt opp på retrospeltiv-møtene). Tidligere så har teamet ofte glemt hva som var forandringen til neste gang.

- *Er det noen andre ting du har lyst til å trekke frem som dere har endret på siden dere begynte med Scrum for 2,5år siden?*

Vi er mye strengere på kvaliteten på backlogen nå enn (vi var) tidligere. Kanskje ikke *mye* strengere. Vi er strengere på det at vi skal ha den mer "tygd inn". Da mener jeg at vi ønsker at det skal være klare akseptansekriterier på user storiene.

Er det product owner som definerer disse akseptansekravene?

Det er han som har ansvar for det. Det vi har gjort er at mellom to sprinter så har *product owner* fått tak på teamet til å være med å lage backlogen. Da har de ikke fungert som et team. Da har de vært hans konsulenter på en måte. På den måten så har vi kjøpt oss litt tid, slik at produktbacklogen er i bedre stand enn den har vært. Du kan godt si at det er galt i forhold til Scrum at vi må samles for hver sprint, men vi har så lang backlog og må få mer og mer på plass. Så har vi også kjørt et kurs på *product ownerne* her og fått sertifisert dem.

Det er nå i senere tid at product ownerne har blitt sertifisert?

Ja, det er helt nylig. Det er 2-3 uker siden. Det har vært *product ownerne* som har vært det største problemet her og det er ikke fordi de er dårlige, men fordi de ikke har vært skolert i Scrum). Det (*product owner*-rollen) er en rolle vi har glemt litt ut.

- *Dere har også endret sprint-lengden, kan du si litt om grunnen til det?*

Det er fordi det kommer inn en del forstyrrelser fra vi starter som fremdeles er problemet. Det kan være f.eks at en person må ut av teamet, "det her må gjøres" eller det er en bugfix. Når vi har 2 uker (sprint-lengde på 2 uker) så er det større sjanse for at vi kan si dette her

må vente til vi er ferdig. Kanskje det bare er en tredel igjen av sprinten, og da kan du utsette det mye lettere ann da vi hadde 4-ukers sprinter. Det er også en sak at vi ønsker å trene litt mer på å gjøre det riktig. Den største fordelen er at du kan utsette ting til neste sprint, i større grad enn før.

Føler du at en kortere sprint-lengde gjør dere mer smidige?

Jeg føler at vi blir mer smidige på den måten. Det anbefales at vi skal gjøre det også. Du oppdager problemer mye tidligere. Før så kunne vi få en stor oppgave som kanskje varte 3 uker, selvfølgelig med underpunkter. Det har vi ikke nå. Det (oppgavene) er enda mer tygd før vi kommer i gang med det. Oppgavene er delt enda mer opp før vi starter med dem.

Jeg har fått høre at dere er begynt med issue-gjennomgang for neste sprint, mens dere er i en sprint. Kan du si litt mer om hva det innebærer?

Ja det er en nokså klassisk ting. Vi skal aldri finne ut av ting og løse dem i samme sprint. Da spiker vi det først, altså prøver å komme nærmere, hva som må gjøres så spinner vi det tilbake til backlogen og så kommer vi tilbake kanskje neste sprint, eller den (sprinten) bak der. Dette er fordi det har vært litt vanskelig å få estimert ting som vi ikke vet hvordan skal gjøres, og samtidig gjøre det i samme sprint. Det er en ganske standard teknikk at du ikke både finner ut hvordan det skal gjøres og gjør det i samme sprint.. hvis man er usikker. Da blir det helt umulig å estimere lengden på det. Da blir det slik at du bruker kanskje 3 story points eller 4-5dager, eller hva du bruker til å måle på, og finner ut hva som må gjøres. Da ser du om det er flere ukers arbeid, eller kanskje en måneds arbeid. Det kunne du ikke estimert før du hadde gjort den spiken. Det anbefaler jeg at flere prøver på også. Ikke estimer ting du ikke vet hvordan skal gjøres.

- *Når det gjelder de forskjellige team-medlemmene har de fått noen spesiell opplæring i Scrum?*

Hele organisasjonen bak hadde et Scrum-kurs, og da var alle på Scrum-kurs. Det var en som het Nils Westergård (som ledet det). Så alle har vært innom Scrum. Alle de ansatte. De nyansatte må dessverre bare hoppe inn i det, men da er det *scrum masters* oppgave å forklare hva vi gjør og hvorfor vi gjør det. Det er litt *scrum masters* coach-oppgave å få forklart dette.

- *Hva ser du på som den viktigste oppgaven du har som scrum master?*

Fjerne impediments.

Hvordan fungerer det å fasilitere at teamene skal være selv-organiserende, å innta en "coachende" rolle..?

Det er litt snedig. For det første så er det, som sagt, å fjerne impediments når det gjelder praktiske problemer og slike ting. Så det kan være like mye å prøve å få folk til gjøre de riktige valgene. Det er litt udefinert hvordan man gjør det. Det kan være måten man hjelper til med å dele opp oppgaver på. Jeg er veldig glad i at folk arbeider i par. Jeg har en idé om at man unngår mange blindveier som et par. Man gjør bare ikke slike feil. Hvis jeg har den tekniske kunnskapen som (skal til), så prøver jeg å prate..men gjerne også å sette sammen nye måter, prate om det, bruke tavla, prøve å skaffe en ekspert og hjelpe til ved å kaste ball med hverandre. Men jeg prøver å unngå å gjøre slik som psykologen og si; "men hva mener du?" og dermed bare kaste ballen tilbake. Men det hjelper jo også for noen. Det viktigste er å få i gang flere til å prate om det (det aktuelle problemet). Der syns jeg at par-arbeid er bra. På små ting så har vi faktisk kastet bort litt tid på møter så vi har heller litt knowledge sharing. Vi tar det gjerne etter den daglige stand-up'en. Så diskuterer vi problemer som folk har. Det aller beste syns jeg faktisk er at noen bryter inn i Scrum'en (det daglige stand-up møtet) for da skjønner jeg at noen har løsning på dette her. Da stopper vi heller den daglige stand-up'en. Da blir jeg i godt humør for da kan du faktisk sette opp et møte etterpå.

- *Du var innom dette med par programmering eller det å arbeide sammen i par, er det andre typiske teknikker som er hentet fra XP som dere benytter dere av?*

Vi prøver å ha test-dreven development eller behavior-driven development, jeg er litt opptatt av at ting heter det de skal hete. Jeg synes det er en del av de viktigste feedback cycle'ene at du driver par programmering. Da syns jeg at det er litt viktig at du spesifiserer hva det er du skal gjøre. Andre grep fra XP er akseptanse-testing av user stories. Det har vi jobbet for å få i gang, men vi er ikke så flinke til det som vi burde ha vært. Det har også litt med *product owner* (å gjøre). Derfor har disse vært på kurs og sett på akseptanse-kriterier for alle stories. Dette må vi vite for å vite om story'en er fullført. Det går ut på at du sier hva som skal til (for at en user story skal være ferdig). Det er teknikker på dette her med user cards. Ofte blir dette til automatiske tester, ikke ofte, kanskje alt for sjelden, men det

burde bli det (user stories burde bli testet opp mot akseptansekristerier ved hjelp av automatiske tester). For det er hensikten (med den ferdigstilte oppgaven) fra teamet til *product owner*, at du har gjort det *product owner* har krevd. Vi prøver få i gang continuous integration, i litt større grad enn for rundt 6 måneder siden. Flere tester skal kjøres før du sjekker inn koden.

Jeg vil heller ikke ha det slik at man setter sammen parene på en bestemt måte, for vi er nokså små grupper. Jeg vil ikke at vi bytter par hver tredje kvarter, at man bytter plasser eller sitter sammen som ”master og slave”. Det at alle skal ha sittet i par med hverandre syns jeg bare er tull, fordi mange sliter med å jobbe sammen i par. Hos oss sitter vi nokså nært. Vi snakker om ting og jobber gjerne i par når vi begynner på oppgaven og så sitter vi én meter unna personen.

Når to personer skal jobbe sammen som et par på en oppgave, hvordan finner de ut hvem de skal jobbe sammen med? Blir de enige seg i mellom eller er det scrum master som setter dem sammen?

Det er best at teamet fordeler dette. Vi setter oppgavene opp på tavlen, som er der vi skal plukke (oppgavene) fra, ideelt sett. Men jeg er jo med på å sette sammen folk her. I stor grad så bestemmer folk det selv. Jeg er ikke sikker på all teori om hvordan jeg skal sette sammen par, men teamene er så små at alle får jobbet i par med hverandre. På teamene her i Stavanger så er vi ikke mer enn 5stk på noen team. Hadde vi hatt team på 15-20 (personer i hvert team) så måtte vi ha satt sammen en teori på hvem som skulle satt sammen i par og hvordan dette skulle gjøres best for å spre kompetansen. Teamene er så små at det skjer av seg selv. Det er jo også slik nye folk som kommer til får opplæring, altså ved å jobbe i par med ulike flinke folk i teamet.

Andre ting som *scrum mastere* har begynt å fokusere på den siste tiden er at vi begynner på toppen av tavlen/taskboarden. Det vil si at alle jobber på samme story, hvis mulig, altså på forskjellige tasker på den story’en. Vi hadde et problem med at vi av og til hadde folk som var halvferdige med arbeidet. På den måten så kan vi garantere at noen (stories) blir helt ferdig. Det er en ting vi har strammet inn på.

Jeg tror faktisk det er par programmering vi har fått mest klager på. Det som folk synes er tungt, det er par programmeringen. Det er slitsomt å sitte sammen med folk å konsentrere

seg og bli uenig med dem som sitter ved siden av deg og du har din rytme på når du surfer, går i kantinen og den typen ting. Det er dessuten tungt å byttte hele tiden.

Vi har ikke kjørt noen personlighetsprofiler på folk som har jobbet sammen i team, så folk blir ikke satt sammen utfra det.

En av de andre scrum masterene snakke også om at teamene var mer isolerte tidligere, der ett team jobbet utelukkene på f.eks modulen maintenance, mens dere prøver å spre dette litt mer nå..

Ja, vi har satt sammen rammeverks-teamet og noe som heter ”Data Management”-teamet. Det (sistnevnte) er et nytt produkt som vi har begynt på. Nå har vi gjerne flere ting på samme team. To *product ownere* går sammen for samme team. De har bare én *product owner*, men det er to *product ownere* bak scenen som steller med hva som skal gjøres. I det tilfellet så er det en egen (*product owner*) for Framework og en *product owner* for ”Data Management”-delen så blir de (de to *product ownerne*) enige inne på kammerset om hva som er viktig, men det er *product owner* for ”Data Management”-delen som fungerer som *product owner* for begge deler. Men vi deler teamet. Det er flere grunner til det. Vi tror at det fungerer bedre på den måten, altså ved at de (team-medlemmene) lærer flere ting (lærer om flere produkter/moduler dvs. f.eks Framework- og ”Data Management”-delen samtidig). Samtidig er det jo nyttig for oss at folk kan gjøre flere ting, for da kan vi sette enda mere ressurser inn på et område som trenger det. Hvordan dette fungerer i Kualulumpur (kontoret i Malaysia) er jeg ikke helt sikker på, men det vet en av de andre *scrum masterne*. Men det er slik at Production(-teamet) og Maintenance-teamet skal slås sammen. Der har vi også fått lokale *scrum mastere*. Før hadde vi *scrum master* her, og teamet i Kualulumpur. Det var ikke ideelt. Spesielt når det gjelder nye ting må *scrum master* kunne være tilgjengelig hele tiden. Men vi mistet de fine turene til Kualulumpur. Men det var ikke motivasjonen, for å spare penger altså. Det var for at vi måtte få *scrum master* nærmere. Det er kanskje et punkt som skiller oss fra mange andre bedrifter, det at vi har team på forskjellige lokaliteter og team der team-medlemmene sitter på forskjellige plasser, som kanskje er det aller vanskeligste.

- *Har dere noen spesielle retningslinjer i forbindelse med dokumentasjon, med tanke på for eksempel når og hvor mye dere skal dokumentere?*

Jeg er litt usikker på hvilket nivå du legger dokumentasjon på. I koden, der har vi klare standarder på det. De kan vi ikke forandre på nå. Det er standard javadoc. Så har vi etterhvert begynt å bruke to ulike måter; vi skal ha en såkalt user documentation og en referanse documentation. Alt det her er det *product owner* som sier ifra om er godt nok. Men alt skal dokumenteres i utgangspunktet. Vi har faktisk hatt folk som har gått rundt på arbeidet og spurt om dokumentasjon, for å sette fokus på det. Det er noen som ikke er i teamet som har gått rundt og gjerne vil se på dokumentasjonen, for å få fokus på det. Men produktet er godt dokumentert. Det er det sikkert 10 000 sider dokumentasjon. Det er egentlig alt for mye.

Grunnen til at jeg tar det opp er jo at Scrum som metodikk har jo et stort fokus på at det ikke skal noe unødvendig dokumentasjon, og da tenker jeg at det kan være vanskelig å vite når du har for mye og når du har for lite...

Scrum er jo det som er management-tool'et, så det er veldig lite dokumentasjon på prosessen, det er "aldri" noe dokumentasjon fra det som kommer fra XP. Det vi gjør når vi dokumenterer på selve Scrum-biten, det er taskboardet. Den dokumenterer vi. Ett bildet av tavlen hver dag og det er faktisk godt nok. Burndown-chartet på tavlen, det tas bildet av. Så har vi startet en blog på de fleste teamene. Der skriver team-medlemmene hva de gjør, problemer de har og møter de har osv. Dette blir tag'et med forskjellige ting. Grunnen til det er at det er et lavterskelt tilbud. Alle tør å skrive på en blog. På en wiki eller lignende så skal folk være så utrolig nøye, forsiktige og formelle at det kommer aldri ut. På teamene har de en blog, ikke en wiki, for da kan du sortere på nøkkelord, på datoen og slike ting. Det er liksom ikke et leksikon. Du søker ikke på nøkkelord og vil se fasiten. Du vil se hva folk gjorde på møter, små figurer, bilder av tavlen og slike ting.

Men det kanskje viktigste er at i Scrum så snakker vi ikke om å dokumentere prosessen. Da har vi selfølgelig en backlog også. Den er litt spesiell. Den er litt vanskelig å finne, men den er der. Jeg kan ikke se hva som kommer om 6 sprinter på backlogen. Det er ikke bestandig at den er klar.

Så tror jeg at i dokumentasjonen innad i XP så er det ganske lite. Vi tar kun vare på de UML-diagrammene fra det ferdige produktet. Vi tegner en del UML mens vi jobber og

prater, men det er på en tavle. Det har ikke betydning utenom mens vi jobber med det. Spesifikasjonen der er gjerne, hvertfall på Unit nivå, på tester, da bruker vi camel case og skriver lange setninger slik at det skal være mulig å lese det som en engelsk setning. Som et sett av spesifikasjoner. Eksempelvis: ”skalLåseKontoenEtterTreInnlogginger”, ”skalVæreMuligÅLoggeInn” osv. Tester kan være dokumentasjon på enhetsnivå. Det er jo, som jeg har nevnt tidligere, der XP-ting som kodestandard, javadoc og den typen ting kommer inn. Når det gjelder produktet, så klarer jeg verken å plasserer det i Scrum eller XP, men så skal det være egne manualer til kunden, og det er gjerne egne tasks at du skal lage manualer.

- *Vi kan gjerne snakke litt mer om review- og retrospektiv-møtene. I reviewmøtene så er det typisk scrum master, product owner og teamet som er med. Er det noen andre som pleier å delta på disse?*

Du kan godt si at release manager er litt som en vaktbikkje for rammeverkene. Det sendes innbydelse til noen få til. Jeg skulle ønske at alle kunne få innbydelse og få lov til å gå på det, men det blir litt vanskelig når vi har 14-dagers sprinter og da blir det så mange demoer. Dessuten kreves det også at folk følger Scrum, at de klarer å holde munn til vi er ferdig og kommer med spørsmål på slutten (i stedet).

I retrospektivene så er det viktigste at hvert teammedlem forteller hvordan de har hatt det, at vi ser tilbake på det. Men også at vi klarer å finne ut disse klassiske spørsmålene: ”Hva gikk bra?”, ”Hva gikk dårlig”, ”Hvordan skal vi fokusere bedre?”. Så tegner vi energi-kurver. Da må vi prøve å lese ut om det hver enkelt føler, er noe subjektivt eller om det har noe med hele teamet å gjøre eller med hele prosessen å gjøre. Det er mye enklere når du har 14-dagers å sprinter å kjøre retrospektiver, for da husker du hva som har skjedd mye bedre. Men det kan fort bli et supperåd for retrospektiver. Det var derfor vi innførte det med å klistre opp ulike lapper på tavlen. Da må du også tegne med en penn mens du snakker slik at du ikke kan lyge. Det er nemlig nesten umulig å tegne en slik kurve mens du ikke prater sant. Du kan bare prøve. Fortell kordan du har hatt det og så tegner du på tavlen mens du lyver. Dette har jeg stjålet fra et konsulentselskap. Det er en helt vanlig måte å gjøre det på.

Vi prøver å ikke bruke mer (tid) på demoer (review-møter) eller retrospektiver enn én time pr. uke, men det klarer vi ikke helt. Vi bruker litt mer. Da gjør vi ferdig demoen og

retrospektiven på to timer. Da vi hadde 4 uker (4-ukers sprinter), så klarte vi det på fire timer.

Når det gjelder sprintplanleggingen, hvor lang tid bruker dere vanligvis på å planlegge en sprint, nå som dere har såpass korte sprinter?

Vi bruker om lag én time pr. uke, slik vi har gjort før. Men vi tror det avhenger av hvor entydige storiene er. Hvis vi må sette på nye akseptansekrav så kan det lett gå mye tid. Vi har et todelt møte. Først med *product owner* som gjennomgår storien og diskuterer, så prøver vi å spesifisere og da trenger ikke *product owner* å være med. Men han må være der helt på slutten, slik at han kan se at han er enig i det som skal gjøres, at vi kan ta handshake og si at det her tar vi. Det er av og til at denne storien må ut og denne må inn og man må gjøre endringer på backlogen. Ideelt sett, på en god backlog, så må man ikke bruke mer enn én time pr. uke.

- *Det har jo vært en del utskiftninger på teamene, hvor lenge er det vanlig at team har jobbet sammen?*

Vi har faktisk byttet på teamene ca. én gang i måneden helt til etter jul. Det har også litt med at vi slo sammen disse teamene slik at teamet skulle dekke flere områder. Da ble det en del skifter og flytting frem og tilbake.

Velocity og focus factor - disse har vært helt umulig å beregne siden det har vært så mye utskiftning. Vi estimerer i story points og vi har data til å lage focus factor. ca. 0,75 story points pr. dag pr. person for de teamene jeg har (er *scrum master* for). 0,75 (story points pr. dag pr. person) for det ene og 0,73 (story points pr. dag pr. person) for det andre.

- *Vi har såvidt vært innom dette tidligere, men er det noen spesielle grep eller teknikker som dere bruker i bedriften som ikke tilhører rammeverket Scrum, men som dere føler må til for at det skal gå rundt eller for at det skal fungere, ting som dere har måttet legge til eller trekke fra rammeverket?*

Vi avbryter ikke sprinter så ofte som det kanskje burde vært gjort, fordi ledelsen dikterer ting; ta folk ut, ta folk inn. En del sprinter burde avbrutt fordi vi endret scopet, men det har vi sett gjennom fingrene på. Så har vi slitt litt med akseptansekravene på noen user stories.

De har vært litt udefinerbare..?

Ja, i stedet for å bruke et par dager på å lage et akseptansekrav, så har vi bare gjort det (begynt å jobbe på oppgaven) fordi vi var nesten sikre på hvordan det skulle gjøres.

En siste ting; I ett teamene har vi en referansegruppe som strengt tatt er stakeholders som burde vært feiet unna, men som får lov å være med å gi gode råd. Det er et fast møte én gang i uken. Vi holder på å lage et nytt rammeverk og da har vi en del eksperter som sitter i en gruppe som tar beslutninger underveis. Strengt tatt så burde det vært *product owner* som hadde denne gruppen her..

Product owner burde vært representert i gruppen..?

Vanligvis skal teamet kun forholde seg til *product owner* under planlegging og til *scrum master* underveis. Men her får teamet møte med en referansegruppe, som strengt tatt er stakeholders, én gang i uken der de tar beslutningene. Det er så mye detaljer og så mange vanskelige spørsmål at vi må nesten gjøre det slik for å komme videre. Det er helt i mot Scrum, men vi synes at det passer på disse tingene.

Hva er typiske saker som blir tatt opp på møter med referansegruppen?

Som sagt, vi skal gå over fra vårt eget rammeverk til et rammeverk som... Det kan være hva du skal ha som klientside-validering og hva skal være serverside-validering. ”Hvor mye javascript kan vi ha?” Det kan vise seg at vi skal tilfredstille kunder i Afrika via satelittelefon, (det kan være) utrolig lang delay på det der, da er det dumt å ha serverside-validering. Vi har en del ting som er helt umulige å få til i det nåværende rammeverket, syns vi hvertfall. Blant annet at valgte felter er gule, mens når du starter å skrive i dem så blir de hvite. Vi vil gjerne ha det slik at du skriver til alle og så kan du peke på de som er der, men de står markert som mandatory hele tiden. Det er slike ting vi må ta beslutninger på hvordan det skal være. Noen tabeller er ikke som vi vil ha det nå. ”Hvilken tabell-komponent bør vi bruke?” Så legger vi frem 6-7 elementer. Så skal de svare ”ja” eller ”nei”, eller si hva vi skal gjøre. Det (referansegruppen) er også et senior-team. Det er de utvalgte menn og kvinner som er med i dette teamet. Det er når teamet (som blir veiledet av referansegruppen) har begynt å få kontroll på hvordan det her skal være at tar vi det ut i andre team og forklarer og får andre folk til å gjøre det (de endringene som har blitt vedtatt i første omgang for dette ene teamet som får hjelp av referansegruppen).

Det er bare ett team som får hjelp av denne referansegruppen?

Ja, og de gjør også en kvirisk oppgave og det skal være for senior-teamet.

Er dette en måte gjøre det på som dere kan videreføre til andre team eller er referansegruppen noe som bare vil kjøres for dette ene teamet?

Det er i slike situasjoner hvor det er vanskelige spørsmål som ikke *product owner* kan svare på alene, som er litt sånn politiske svar, så vurderer vi det av og til. Referense-gruppene består av ressurspersoner som kanskje kan brukes som domeneeksperter eller som du bare kun får tak på i noen timer i uken.

- *Er det noe du synes fungerer spesielt bra med hvordan dere kjører Scrum på, noe som skiller seg ut som dere får til å fungere veldig bra?*

Hvis jeg skal si noe, så vil jeg si at vi er flinkest på retrospektivene. Der har jeg vært inni en del andre bedrifter og sett hvordan de gjør det og da har det vært litt supperåd. Jeg er også litt stolt over at vi har klart å fjerne oss fra såpass mye typer verktøy for å holde styr på backlog og daglig drift med regneark og ressurser. Vi bruker bare en tavle og tar bilde av det. Tidligere var det så mye regneark-styr som skulle vedlikeholdes og plottes inn.

Når det gjelder å prioritere forskjellige items i produkt backlogen så fikk jeg inntrykk fra én av de andre scrum masterne at det er ganske vanskelig å få dette til i Jira og i den forbindelse var det snakk om at dere var i ferd med å utvikle et internt verktøy for dette..?

Det har vi tenkt litt på. Vi lurt på det lenge. Da jeg var i Kualalumpur så syntes sjefen for kontoret der at det var litt dumt at vi skulle bruke tavle, for vanligvis laget vi jo dataprogrammer for alt, alt fra gule lapper og det ballet ofte på seg. Men vi trenger en del ting for i produktbacklogen ligger det user stories, i Jira ligger det oppgaver, tasks, dokumenter og bugs osv. Så det er ikke en én-til-én kobling mellom dem. En task kan være flere fra Jira osv. En Jira kan gi flere tasks. Du kan si at bugs og bokholderi og arkivet er i Jira, men vi ønsker et verktøy som det er lettere å lage sprinter av og er lettere å jobbe med. Det må være en kobling, men hvordan det blir det vet jeg ikke. Det er også noen som ønsker at det skal være litt tilbake til justering av sprinter osv, fordi ledelsen vil kunne se alle bug'ene samtidig ved bare å åpne en webside. Vi får se, men det er litt avhengig av hvor mye måling og veing vi skal ha. Vi er litt uenig. Noen syns det er viktig å telle mål og

veie, mens andre som meg syns ikke det. Vi syns ikke vi skal telle mål og veie inne i sprinten vi vil heller telle outputen av sprintene. Hvor mye et team klarer å produsere. Vi syns ikke det er så viktig å se på dette fra dag til dag. Da syns jeg at burndown og å tegne på tavlen er godt nok. Men der er vi uenige. Det får vi se på.

Vi har sett på eksisterende verktøy også. Et som heter ScrumWorks som er bra på noen ting, men dyrt. Det er litt for stort også.

I forbindelse med planleggingen av en sprint, går product owner da inn i Jira og velger ut oppgaver blant dem som ligger der inne og prioriterer utfra hva som ligger inne i Jira?

Nå har det fungert som et regneark. Det har vært slik at man har plukket fra Jira. I Jira kommer ting inn og så går man gjennom ting og ser hva som skal ut og setter det på den backlogen der. Akkurat den koblingen fra Jira til produkt backlog er vanskelig. Det kunne vært interessant å se i backlogen hva som skjer i de ulike inkrementene, også historisk sett. Man må også ha en backlog der en kan spå hva som kan leveres om et halvt år.

Det er koblingen som er litt vanskelig. Vi har ikke funnet en perfekt kobling i ScrumWorks. Det var ikke helt ideelt. Men vi ønsker også å ha et verktøy som støtter den templaten vi skal ha for user card med type; ”As a *product owner* I want to..” Mer slike ting. Å sette opp akseptansekristerier og at du lett kan sjonglere med disse tingene. Vi trenger også en virtuell tavle mot Kualalumpur for task boardet. Da ønsker jeg at det skal faktisk se ut som en tavle og at det faktisk henger gule lapper på den.

Benytter dere mulighetene videokonferanse gir mot Kualalumpur?

Hvis du regner Office Communicator med et web camera som en videokonferanse så gjør vi det. Problemet er det at linjene der nede er ikke all verden. Det går på (Office) Communicator og telefonkonferansesystem.

Det er to team hvor teammedlemmer er spredt over både Norge og i Malaysia (Kualalumpur), og når de skal ha daily stand-up meeting hvordan foregår da det?

Da har teamet et felles ”vindu”. Når klokken er tre (kl.15:00) der (i Kualalumpur) så er den ni (kl.09:00) her. Men nå er det slik at når den er fire (kl.16:00) der, så er den ni (kl.09:00) her. De starter litt sent og jobber lenger der. Jeg har vært der selv og det er sjelden man er på jobb før ti (kl.10:00) der. Det er så mye kø på motorveien også der. Det (Kualalumpur-

kontoret) ligger midt i en teknopark som er bygget opp for å tiltrekke seg vestlige firmaer. Men den ligger, selvfølgelig, dårlig plassert i forhold til flyplassen. Det er jo også kjekt at de går på jobb først, for da har vi faktisk 15-timers arbeidsdag. Da begynner de på arbeidsoppgaver og så setter de dem over til oss etterpå. Når ting brenner skikkelig, så er det faktisk godt at vi har denne tidsforskjellen. Det er også litt morsomt, for de har stand-up (meeting) på slutten av dagen og vi har på begynnelsen (av dagen). Jeg syns i grunn det er best å ha stand-up tidlig på dagen, for da husker du bedre. Man kan si at ulempen ved å ha det (stand-up møte) tidlig på dagen er at da glemmer du hva du gjorde i går, men jeg syns faktisk det er verre at når du skal ha stand-up'en og si hva du skal gjøre dagen etterpå og så glemmer du det dagen etterpå - hva du skal gjøre egentlig. Derfor liker jeg best å ha det først på dagen.

B.4 Interview with chief architect in project-12, 11.March 09

❖ *Hvilke utfordringer ser du i forbindelse med prosjektledelse (innenfor Scrum)?*

Det som er vanskelig for en *scrum master* er å ha riktig balanse mellom å støtte og kontrollere, slik at teamet tar regnskap i avgjørelsene samtidig som at avgjørelsene blir tatt.

Det viktigste og vanskeligste å få til er en god product backlog. Vi har sett at der hvor man klarer å ha en saklig dialog med *product owner* rundt en product backlog og hvor man klarer å.. Vi har to backloger, altså to køer. Sprint backlogene pleier som regel teamene å klare sånn tålig bra, og når de ikke gjør det så er det ikke noen katastrofe. En dårlig product backlog (derimot) gjør at vi ikke lykkes.

Det er veldig vanlig på de prosjektene jeg har sett at *product ownere* har mangelfull opplæring eller den som av andre grunner er *product owner*, ikke har kompetanse på det området.

Hvordan kan product owner få riktig kompetanse hvis han stiller på bar bakke og hvordan skal han klare å vedlikeholde produdt backlogen?

Det åpenbare svaret vil da være enten opplæring eller lese litteratur. Problemet er at de som ofte er gode *product owners* er ofte dem som har veldig lite tid. Så det er en liten catch der, fordi du har en person som er *product owner* fordi han har en viktig stilling innenfor den organisasjonen han er i, og han føler ikke at han har tid til å gå på kurs, men han har ikke

den kompetansen som skal til for å gjøre jobben optimalt. Det som er viktig da, hvis jeg skal gi et råd.. *Product owners* ansvar er jo at systemene leverer verdi. Så det som er det lette svaret, som nødvendigvis ikke hjelper så mye, er at *product owner* må fokusere på å forstå verdien og formidle verdien til de som skal lage systemet og ikke minst, det er det som gjør det veldig vanskelig, tror jeg, å klare å tenke på kortsiktig verdi såvel som langsiktig verdi. Det er den kortsiktige verdien som folk sliter med. Folk vet gjerne hvordan de vil at ting skal være om to år, men ikke hvordan de vil at det skal være om tre måneder.

Har dere en spesiell måte dere lærer opp nye product owners på innad i bedriften? Sender dere de på et spesielt kurs eller er det intern opplæring?

I vår bedrift er det som regel slik at kunden skal være *product owner*, så det (opplæring) er på mange måter ikke vårt ansvar.

Har det vært slik hele siden eller er dette noe som har endret seg?

Det har alltid vært den vanlige antakelsen at det er det vi ønsker. Nå er det noe med at man har en prosjektleder eller en klasse prosjekteledere eller en analytiker som tar rollen for vårt vedkommende, spesielt på det prosjektet vi er på nå, som er ganske stort, så har vi noen analytikere fra vår side som tar det ansvaret. De har ikke det samme problemet med kompetansen, men de har jo da problemet med eierskapet og myndigheten for driften.

Å kommunisere med kunden blir jo da veldig viktig. Gjør dere noen spesielle tiltak for å oppnå dette eller er det en spesiell måte dere griper dette an på?

Det vi har nå er et prosjekt som er på 30 personer, alt inkludert, på vår side og da er det ca. to fulltidsressurser som hele tiden jobber med de ekte *product ownerne* med product backlog. I tillegg er det to personer (fulltidsressurser) fra kunden også. Dette prosjektet har pågått i rundt to måneder, og der har ikke ting stabilisert seg nok enda til at vi kan bruke mindre ressurser, det kan hende at det går seg til etterhvert. Det er mye ny mark som må pløyes.

Jeg tenker på din rolle oppi dette. Er du utvikler hovedsaklig?

Min tittel i prosjektet er chief architect, men jeg tror at det bare er en utvikler som får litt ekstra betalt. Jeg har også litt opplæringsansvar for *scrum masterne* spesielt.

Har du jobbet mye som scrum master tidligere?

På det forrige prosjektet så hadde vi uoffisielt Scrum og da var jeg team lead på det. På et forholdsvis lengre prosjekt, for den forrige bedriften jeg jobbet for, så var jeg uoffisielt *scrum master* siden vi da ikke hadde en offisiell prosess. Vi kjørte prosessen ca. 3 måneder, kanskje litt mer. Etter det så var jeg også kort *scrum master* på et annet prosjekt for den bedriften jeg jobber i nå. Det var kun et én-måneds-opplegg, så det var mer i overgangsfasen. Jeg har faktisk aldri offisielt sett hatt "*scrum master*"- tittelen.

Har du tatt scrum master sertifisering eller hatt noen form for spesiell trening?

Forsåvidt, men det var lenge etter at jeg kunne Scrum. Jeg har holdt et halvt *scrum master* kurs også, så jeg har vært litt på begge sider. Det var da jeg diskuterte med Mike Cohn hva som definerer Scrum, eller ikke.

På det prosjektet (som han nevnte tidligere for den forrige bedriften han jobbet i) så var jeg team lead/*scrum master* i rundt halvannet år.

- *Hvis vi ser på din rolle som hovedsaklig utvikler, hva ser du på som din viktigste oppgave som utvikler under en Scrum? Under den som du er en del av nå for eksempel...*

Det som er arkitekt eller en utvikler eller tester sin rolle er jo en del av den rollen som kalles teamet, og teamet har som jobb å levere kvalitet. Jobben er å gjøre det som skal til for at det som står på sprint backlogen blir ferdigstilt. Det som veldig gjerne kjennetegner et Scrum team er at alle må ta litt mer ansvar. Det vil si at en utvikler kan ikke si at nå er jeg utvikler så da er det tester sitt problem, og tilsvarende kan ikke en tester si at utviklerne har ikke kommet opp med et spesielt verktøy, så da sitter jeg bare her og venter på det. Så rollene blir ganske løst definerte. Det tror er noe av grunnen til at man snakker om én rolle som er teamet, og ikke utvikler, tester osv.

Når det gjelder for eksempel det prosjektet dere er inne på nå, så er det kanskje ikke alle som er med på prosjektet som har like mye erfaring med Scrum fra før, men har dere noen spesielle tiltak for å lære opp for eksempel andre utviklere eller folk som er med på prosjektet?

Det som er veldig viktige tiltak som vi gjør generelt sett, er det å sørge for at folk jobber mye sammen og det gjelder opplæring både på Scrum, det gjelder opplæring på teknologien, på utviklingsmetoder og den typen ting. På våre to team nå så vil jeg anslå at i snitt så jobber folk med par-programmering eller tilsvarende rundt tre timer daglig. Noen mer, noen mindre. Mye av de metodene vi følger fra Scrum, retrospektiv og stand-up møtet spesielt er jo veldig fokusert på det å spre kunnskapen.

Er det noen andre typiske teknikker fra XP utenom par-programmering som dere praktiserer?

Vi bruker aktivt test-drevet utviking. Vi bruker continous integration. Vi fokuserer på små leveranser, men det er jo alltid en utfordring. Jeg tror vi har små nok, men vi må nok forandre på ting. Vi har forsøkt å styre mot ”simple design”, men det er jo en vurderingssak om man har oppnådd det eller ikke. Du kan si at vi tok bort all det vi hadde av unødvendige arkitektur-”ræl” i stakken og så tok vi bort alt det som man følte man kunne ta bort og så tok vi bort litt til, for å være helt sikker. Så vi har kuttet til beinet på alt som skal til for å løse oppgavene våre. Refactoring også naturligvis, men det faller inn under samme bolken.

Når det gjelder par-programmering, er det da noen spesielle måter dere gjør det på? For eksempel at dere bytter mellom parene eller er det noen som alltid ”teamer opp”, noe dere gjør for å legge opp til mer kunnskapsspredning?

På ett team så har man gjort noe aktivt i forhold til det. Hvor man sier; ”du jobbet med han i går, så nå går du kanskje heller til noen andre.” Men når det gjelder de andre teamene så er det mindre bevisste prosesser rundt det.

Finnes det noen fastlagte retningslinjer for at projektet bør bruke par-programmering eller er det opp til hvert team i hvilken grad de vil benytte det?

Jeg tror det er et ønske fra prosjektet, men det er ikke et krav og det var teamene selv som da sa at de skulle prøve det. Det teamet som har gjort det best sa; ”vi skal par programmere en del og vi vet ikke helt hvor mye en del er.” De som har gjort det litt mindre bra, sa at vi vil prøve å par programmere 2 timer daglig i snitt pr. person. Så det var teamene som sa dette. Da vi gikk gjennom hva vi skulle ha som arbeidsregler så var et av punktene *scrum master* gikk gjennom; skal vi par programmere, og i såfall i hvilken grad?

❖ *Hvilken sprintlengde bruker dere på det prosjektet dere jobber på nå?*

Vi bruker tre uker. Det er det som er påkrevd av organisasjonen. Prosjektet har synkroniserte sprinter over ti team nå, tror jeg.

Har dere noen form for, for eksempel Scrum of Scrums, for å kunne koordinere de ulike teamene?

Ja, men jeg har ikke gått på dem. Jeg tror faktisk Scrum of Scrums er mest teoretisk mulig. Det blir bare ”suppemøter” ut av det.

Så det har ikke blitt praktisert frem til nå?

Det har blitt praktisert. Jeg tror ikke at det fungerer i praksis, men det har blitt forsøkt.

Når dere har kjørt Scrum of Scrums, hvor ofte har disse blitt møtene blitt avholdt?

Ca. tre ganger i uken.

Ser du for deg at det kan være noen substitutter for denne typen møter (Scrum of Scrums)?

Det ene er å sørge for at teamene er uavhengige, eller at teamene er mindre avhengige, da tror jeg at vi reduserer behovet. Det gjør man ved at hvert team kan uavhengig levere verdi. Dette gjør at hvert team har en product backlog som henger sammen. Vi har to team som jeg er involvert i og 8 andre team. Våre to team sitter rett ved siden av hverandre, så vi har mye informasjonsflyt mellom dem, ergo finnes det flere uformelle kanaler. De ser ut til å fungere bedre. Det som vi også har gjort, som er min favoritt metode, er å være chicken.. hvis du kjenner den terminologien..?

Nei, det gjør jeg ikke. Kan du forklare den litt nærmere?

Man bruker en metafor, der du har chicken & pig, hvor du har at; ”the chicken & the pig were starting a breakfast restaurant together. What would we serve? Bacon (ham) and eggs. You would be involved, but I would be committed.” Så kylling er de som ikke er comitted på det prosjektet og som da er på besøk til andre team sine Scrum-møter og er kylling der. Det er ofte et godt substitutt for Scrum of Scrums. Det ungerte rimelig bra på det forrige prosjektet hvor vi hadde stand-up møter. I den forrige bedriften jeg jobbet i

hadde vi stand-up møter fra 2004, tror jeg, og da var jeg kylling på mange av de stand-up møtene for fire (forskjellige) team.

Figure B.3: Illustrasjon av "pig & chicken"-metaforen.

Hvem er det da som typisk er kylling på de andre møtene, er det utviklere, scrum master eller begge deler?

Scrum master primært, og folk som har mer tilsluttende roller som prosjektleder eller arkitekt.

Vi har også en mindre Scrum of Scrums som bare gjelder disse to teamene (som intervjuobjektet har tilknytning til), pluss noen flere personer og det funker ganske greit. Det som er problemet er de brede Scrum of Scrumene hvor folk jobber med så forskjellige ting at de har ikke noen interesse av hverandre.

Da er det team som jobber med de samme oppgavene som holder slike møter?

Ja, oppgaver som er veldig nær hverandre. Disse to teamene som vi jobber med, jobber med samme kodebase så det blir overlappende i den kodebasen, men med litt forskjellige sprint backlog-elementer.

Ja, sprintlengde. Det er en litt interessant historie fra den forrige bedriften jeg jobbet for, som jeg bare har indirekte bekreftelse på. Der kjørte vi 2-ukers sprinter og det syns jeg på en måte fungerte bedre enn 3-uker, som vi kjører nå (i den bedriften han jobber for nå). Det som man har gjort i etterkant, som jeg ikke har vært med på men som jeg har fått referert, er at man kjørte 4-ukers sprinter hvor man var produksjonsklare for hver sprint. Så det er jo et valg man må ta der, om man vil ha en hyppig og grunn sprint eller en lengre og dyp

sprint. Altså, om du skal faktisk levere og må ha lengre sprinter eller om du skal gjøre noe leverbart, men kanskje ikke levere det.

Men nå som dere, i den bedriften du jobber i nå, har 3-ukers sprinter har dere på en måte en slags mellomting. Hva er tanken bak det?

Det som er typisk for en 3-ukers sprint, som du ikke får til på en 2-ukers sprint er at du har produksjonsmiljø eller pre-produksjonsmiljø som er produksjonsnært, men man har ikke faktisk produksjon. Jeg har vel til gode å se et team som leverer til produksjon pr. sprint på tre uker eller mindre. Det er også noe som overses, at man kan ofte velge om man vil ha en grundig sprint eller en kort sprint.

Endrer dere mye (på sprintlengden) fra gang til gang eller holder dere sprintlengden forholdvis stabil?

Ja, veldig stabil. I den forrige bedriften jeg jobbet for hadde vi sprintlengden stabil i to år, tror jeg. Jeg tror det er få team som endrer sprintlengden mer enn én gang i året, av de som jeg har snakket med i hvertfall. Da tenker jeg generelt sett, ikke bare innenfor den bedriften jeg jobber for nå. Når jeg snakker med andre folk som bruker Scrum, så er det typisk at man liker å ha en stabil sprint-lengde. Men så finnes det noen som gjør ”rare” ting når man kommer til påske og jul.

Hva er det som er typisk når det gjelder teamsammensetning, er det stabile team?

I de teamene som vi er inne på nå (i den bedriften han jobber for nå), så har de bare vært i gang i 1-2 måneder. I de andre teamene som jeg har vært involvert i så har de vært stabile inntil 60/70/80 % av tiden. I den forrige bedriften jeg jobbet for så var det typisk stabile team i over ett år. Det betyr at du har noe større turnover i et team enn i organisasjonen som helhet, men ikke betydelig. Så i den forrige bedriften jeg jobbet for, var det oftere at det var en konsulent som gikk ut og dermed forsvant fra teamet enn at det var en som ble overført til et annet team.

Hvor mange er dere typisk i et team?

I den bedriften jeg jobber for nå, på det prosjektet vi er på nå, så er vi 8 personer, tror jeg, det inkluderer funksjonelt ansvarlig og *scrum master*. Vi er 8 på den ene og 9 på det andre av de to teamene vi har der nå. De andre teamene har samme størrelse.

I den forrige bedriften jeg jobbet for, så hadde vi alt fra.. jeg tror det minste var 4, og da var det også noen som var ute i fødselspermisjon, så i praksis så var vi faktisk mindre. I det største (teamet) var vi 16, tror jeg.

Hvordan vil du si at måten dere kjører Scrum på endrer seg i forhold til teamstørrelse?

Det værste problemet er når du har team som er veldig avhengig av hverandre. Det er et større problem, syns jeg, enn de største teamene vi hadde. Team opp til 12 personer fungerte til dels bedre enn team på 6-7 personer som måtte koordinere mye seg i mellom. Når vi er 6-7 (personer) så er det best. 6-7 som kan jobbe uavhengig så pleier det å fungere bra. 8-9 funker greit.

På review- og retrospektive-møtene, vi kan ta review-møtene først, hvem er det som typisk deltar på disse møtene?

I den bedriften jeg jobber for nå er det saksbehandlere, forretningsansvarlig og kunden.

I den forrige bedriften jeg jobbet for var det teamet, *scrum master* og *product owner*.

Endrer dette seg når det gjelder retrospektive-møtet eller er det de samme personene som deltar som var med på review-møtet?

Jeg har kun vært på ett retrospektive-møte hvor *product owner* har vært med.

Hender det at folk fra andre team kommer innom på retrospektivemøtene, slik vi var innepå at folk gjorde på stand-up møter?

Det har jeg ikke opplevd.

Det vi har tenkt å gjøre nå som vi har to team som jobber mye sammen, som er planen i dag som jeg ikke vet om ble gjennomført men jeg regner med det, er at de har hvert sitt retrospektive-møte og så har de et felles retrospektive-møte etterpå. Vi snakket noe om idéen om retrospektive av retrospektive-møter. Jeg har ikke sett det enda. Planen var jo å kjøre det i dag (mens intervjuobjektet er her i Trondheim).

Når det gjelder smidig og Scrum så er det veldig viktig å tilpasse seg hele tiden i forhold til omgivelsene, i forhold til krav som endrer seg. Nå har jo ikke prosjektet du er på nå vart så lenge, men hvor smidig føler du at dere er på dette prosjektet?

På dette prosjektet føler jeg at vi er ganske smidige. Overordnet så er prosjektet en del av en reform og der var det jo slik at på søndag for to uker siden så fikk da prosjektleder vite, via dagsrevyen, at regjeringen hadde endret omfanget på en relevant reform. Det vil si egentlig ikke en reform, men en del av en relevant lov. Så det er noe vi er klar over, noe frustrert over, men som vi tilpasser oss.

Det er et litt vanskelig spørsmål, fordi det kan bety to forskjellige ting. Det er endringer i krav og endringer i organisajonen, arbeidsmessige forhold altså hvordan man skal jobbe. Spesielt den biten som berører endringer i krav var kanskje den jeg svarte på tidligere.

Den forrige bedriften jeg jobbet for endret seg veldig over den tiden hvor vi brukte Scrum. Vi gikk fra en uformell bruk av til en virksomhetsstandard. Underveis endret man veldig mange arbeidsmetoder og teknologivalg osv. Det krevde ganske mye voksesmerter for organisasjonen å gjøre dette. Det å gå bort fra dårlige teknologivalg ble oppfattet som en risikoøkende faktor av enkelte ledere som vi kan si at ikke lenger er i organisasjonen. Det er det jeg mener med voksesmerter.

Kan du være litt konkret når det gjelder tilpasninger dere gjorde og ting dere gikk bort fra i forhold til tidligere?

Teknologisk gikk vi bort fra en Java EE server til en mer effektiv løsning. Vi gikk bort fra WebSphere til å bruke Jetty som applikasjonsserver. Dette var i forbindelse med at vi også endret på arbeidsprosessene. Det gikk to-veis med arbeidsprosesser. Det vi også gjorde var vi gikk bort fra en overleveringsprosess, fra utvikling til deployment, til en prosess hvor utvikling og drift kommuniserte mer direkte. Det viktigste her var å bryte ned kommunikasjonsbarrierene som eksisterte (og overleveringer).

Var det slik at dere, i den forrige bedriften du jobbet for, i en periode kjørte Scrum og et plan-driven approach side om side?

Ja, oppå hverandre kan du si. Jeg tror den beste måten å beskrive det på var at vi ville kjøre Scrum, men vi ville ha en ordentlig test-prosess på slutten. Testerne skal starte med sitt

arbeid for det de skal gjøre, altså begynne å lage testplaner. Det må de starte med parallelt med utviklingen. Testerne er vant til man skriver et stor kravspesifikasjondokument tidlig, som de kan ta inn i arbeidet og så kan man gjerne kjøre iterasjoner, men du sitter jo ganske mye inni en sånn boks (peker på Testing-boksen) som er ganske fastspikret på alle kanter, i og med at krav-spesifikasjonen er ganske fastspikret på forhånd. Da må man kanskje ta flere runder/iterasjoner enn forventet for å komme gjennom krav-spesifikasjonene, og da har man ingen annen mulighet enn å la hele prosjektet skli (utover i tid, slik at det ikke overholder tidsfrister). Dette fordi man ikke har skrevet krav-spesifikasjonen på en måte som tillater delvise leveranser.

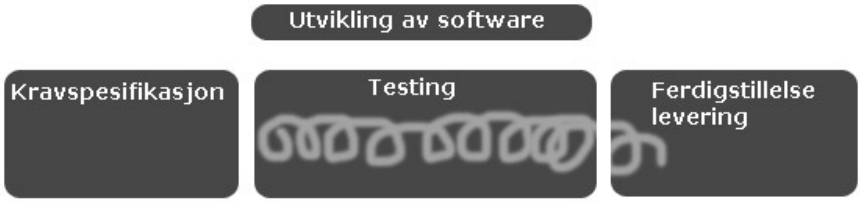

Figure B.4: Infleksibel testprosess som har sklidd utover sin tidsramme.

(På figuren ovenfor ser vi at testprosessen ikke har vært fleksibel nok i forhold til kravspesifikasjonen og har derfor sklidd utover deadline og inn i leveringsfasen.)

Og det var kanskje noe av grunnen til at dere gikk over til Scrum i hovedsak?

Ja! Det var rett og slett det at den plan-drevne prosessen ikke gjorde et prosjekt i stand til å levere. Så man fikk fullført iterasjoner, men ikke levert dem.

Nå har du jo ikke vært i den bedriften du jobber for nå så lenge, men hvordan har du inntrykk av at utviklingen av Scrum som metode har vært der? Men først hvor lenge har de faktisk drevet med Scrum i den bedriften du jobber for nå?

Jeg vet ikke helt sikkert, men over et år hvertfall. Jeg vet ikke utover det. Scrum er jo en prosess som ofte starter med at man innfører det uoffisielt og før du vet ordet av det så er den etabert. Da er det ikke lett å vite når den startet. Det kan jo tenkes at det er folk som har jobbet (med Scrum) mye lengre enn det (1år). Det er da den laveste grensen, men det kan godt tenkes at den høyeste grensen er mye høyere.

Hvordan er det tilsvarende for den forrige bedriften du jobbet for (hvor lenge har de anvendt Scrum)?

Offisielt ble Scrum en preferert metode for ca. 1 år siden. (Scrum ble) i praksis mest brukt for ca. 2 år siden, og det ble begynt innført for 3-4år siden. Så det er veldig mange datoer som virker meningsfylle.

Når du sier "mest brukt for 2 år siden, men foretrukket nå", det er litt spesielt..?

Ja, det er litt spesielt. Det er fordi det er en del (personer) som eier standardene som ikke liker å endre dem. Så prosessene for å endre hva man er enige om er ofte veldig forskjellige fra prosessene for å endre hva man gjør. Ofte tregere. Det er nå 2 år siden jeg ble bedt om å presentere utviklingsmetoden vi brukte i den forrige bedriften jeg jobbet i for kreditttilsynet. Vi viste frem Scrum uten å kalle det Scrum.

Var det en spesiell grunn til at dere gjorde det?

Det var fordi vi ikke kunne, politisk sett i organisasjonen, bruke ordet Scrum. Det for å unngå å lage store bølger.

Er det noe du syns fungerer spesielt bra med slik dere kjører akkurat dette prosjektet (i den bedriften du jobber i nå) i forhold til Scrum og prosjektledelse? Noe dere har lært fra tidligere erfaringer for eksempel...

Det vi har lært er å ha fokus på å kunne levere noe som har potensiell verdi underveis, samt det å ikke glemme de kortsiktige målene blant alle de langsiktige visjonene. Vi har allerede fra første iterasjon laget ting som er funksjonelt brukbare, hvor vi da vet hva vi skal gjøre for å få det til å være en forbedring fra det vi har i dag. Nå er det jo åpenbart ikke en forbedring at vi gjør en syvendedel av det vi er interessert i, men den syvendedelen gjør vi bra.

Da er det vel kanskje viktig å få tilbakemelding fra kunden på den delen dere har produsert på et tidlig tidspunkt for å sjekke at denne stemmer overens med kundens forventninger? Hvordan foregår i såfall denne kommunikasjonen?

Ja. Det som vi har gjort, var at vi inviterte veldig bredt når det gjaldt kundens saksbehandlere for å se på en sprint review. Det var for å få oppnå denne feedbacken. Det

vi ser er at vi må gjøre en del på brukerinnvolvering ved å ha brukertesting, ved å ha beta-brukere, ved å ha testversjoner liggende ute som folk kan benytte seg av og den typen ting.

Dette er veldig spennende. Kan du fortelle litt mer om hvordan beta-testene og brukertestingen foregår eller hvordan dere praktiserer dette?

Vi har i dag et eksisterende system. Vi har saksbehandlere som bruker det systemet og de er ikke fornøyd med det, men de er redde for noe nytt skal bli enda verre. Det tror jeg er en veldig vanlig situasjon. Dette systemet går mot en database. Denne databasen vet vi har såpass mye verdifulle data at vi kan ikke starte på nytt igjen. Det gir en fin begrensning, for det betyr at det nye systemet vårt må gå mot den samme basen. Det vil si at vi må lage et nytt system som fungerer i parallell med det gamle. Før vi har det nye systemet, så ønsker vi å ha en åpen beta-versjon på det. Det vil si at vi rekrutterer noen som vi ser kan være ambassadører og så håper vi at de vil snakke i lunsjen om hvor mye bedre det er å bruke det nye systemet. Det håper vi at vi får brukerne over på et par måneder før vi må si at nå MÅ dere bruke det (det nye systemet).

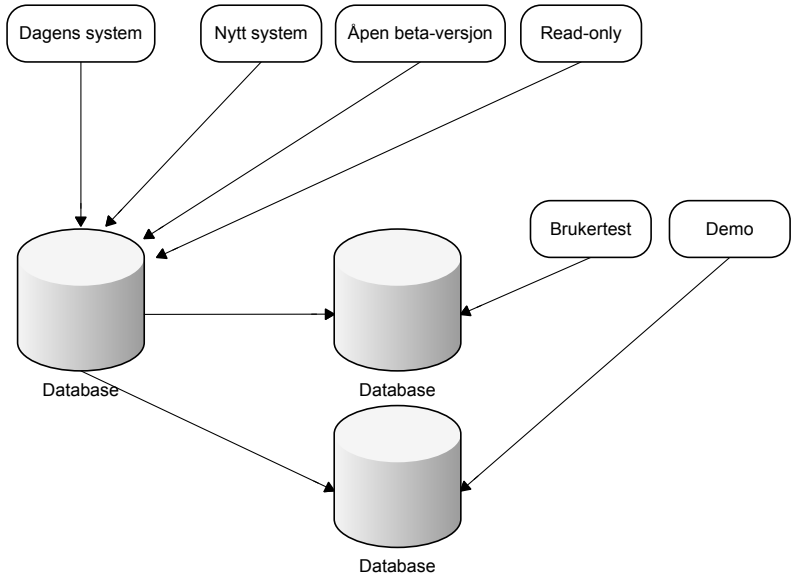

Figure B.5: Testsystemet.

Hvor mange er dere interessert i å få til å bruke den nye åpne beta-versjonen?

Det jeg er interessert i, er å få til at vi fortsetter denne typen syklus når vi får nye (versjoner av systemet), slik at det kommer flere av av disse (utgaver av det nye systemet, peker på ”Åpen beta-versjon”- og ”Read-only”-boksene). Mitt håp er at alle brukerne skal bruke den

åpne betaen inntil de støter på et problem og da går de bare tilbake til det gamle systemet. Det er egentlig en måte å vri seg litt unna et regime hvor du bare kan produksjonsteste noen få ganger i året. Vi har den åpne betaen og så har vi en sak som jeg ikke riktig har et navn på, men er en read-only løsning som er der for å vise brukergrensesnittet. Read-only løsningen kan gå mot produksjonsapparatet. Alle disse går jo mot produksjonssystemet (peker og viser på tegning). Så har vi testsystemet. Vi vil ha brukertesting, brukerlab og demoversjon som går mot en hver sin database. Her kan vi se folk gjøre oppgavene, men de kan ikke skade dataene. Det er da typisk en kopi av deler eller hele den orginale databasen. Vi ønsker i den grad det er mulig å ha en kopi av hele databasen, men det er jo visse plassmessige begrensninger. Vi vil ha noe som vil oppleves som om det var en kopi for deres vedkommende. Forskjellen på brukertesten og demo-versjonen er at, denne (brukertesten) ønsker vi å rulle ut for i hvertfall hver iterasjon, kanskje ved hver bygg.

Formålet med å bytte ut Websphere med Jetty, som nevnt i den forrige bedriften jeg jobbet for, var å kunne rulle ut testversjoner automatisk etter hver bygg, stort sett etter hver innsjekking. Det som skjer etter at jeg har sjekket inn en endring, er at 20-30min senere så er den endringen i drift. Da kjøres den gjennom automatiske tester på systemnivå. Demo-versjonen er den vi ruller ut etter at de (kunden) har vært på review-møte, og da kan de prøve det ut selv. Dette regner jeg med at vi kommer til å få prøvd ut på iterasjon to (av prosjektet).

Det som vi har jobbet mye med i den forrige bedriften jeg jobbet for, er at vi har like mye miljø til en veldig lav-taktet iterasjon. Når dette miljøet har kommet til, er det egentlig en ny kopi av en executable jar på en server som de andre allerede kjører på som er et eget directory, som vi kjører det til.

Vi har folk (i den bedriften han jobber for nå) som jobber med design, interasksjons-design, som drar aktivt med seg fokusgrupper og lar dem bruke det nye systemet og observerer dem. Vi blir enige med dem som er leder for saksbehandlerne at de må plukke ut noen som skal være testbrukerne våre. Da må de sette av et par timer. Vi bør ha to typer brukere. Vi bør ha ambassadørene og skeptikerne.

Jeg er interessert i å få vite litt om tilpasninger som dere har gjort i den bedriften du jobber i nå i forhold til Scrum og endringer de har gjort underveis, de har jo holdt på litt over ett år. Har de f.eks kjørt den samme sprintlengden i løpet av dette året eller har gjort andre tilpasninger når det gjelder opplæring av ansatte eller andre aspekter..

Som et konsulenthus så vil jo bedriftens (den bedriften han jobber i nå) bruk av Scrum være tilpasset hver enkelt kunde. Det vil være store forskjeller. Vi har noen kunder som ønsker å kjøre Scrum og da er det kunden som i hovedsak legger føringene for hva Scrum betyr. Samtidig så har vi også en informasjonspakke, som ikke er publisert, som vi bruker for å formidle Scrum når det er aktuelt. Det som jeg er veldig opptatt av er at vi ikke skal si at vi bruker en variant av Scrum eller vi bruker en tilpasset Scrum. Hvis du sier at Scrum er 3+4+3 (modellen for oppdelingen av Scrum-prosessen). Det er rollene (teamet, *scrum master*, *product owner*), seremoniene (sprint planning, daily stand-up, review, retrospektiv) og artifaktene (product backlog, sprintbacklog, burndown chart). Scrum er også at man tilpasser ting til teamet og prosessen man er i. Det vi har kjørt bevisst på i en del team, 5 team som jeg har vært i kontakt med over 3-4 prosjekter, er at vi legger på en visjon som en ekstra artifakt. En visjon for leveransen. En visjon beskriver et mål som skal oppnås, interessenter og mer detaljer rundt det. Interessentene deler vi opp i to grupper i praksis. Det er personas og prosjekteiere (viktige mennesker på prosjektet).

I tillegg til disse prosjektene her, så har vi brukt dette på et prosjekt rundt dette med å utvikle et Scrum-kurs. Da brukte vi en metodikk som kalles ”pressemelding fra fremtiden.” Da gjør vi som følger: la oss si at dette prosjektet eller dette kurset, ble planlagt og vi fikk gjennomført det og hvordan kunne en pressemelding se ut. Da har vi at kurspartneren kommer med en uttalelse om hvor mye penger det var (som ble brukt). Administrerende direktør kommer med en uttalelse om hvor viktig det var å spre den kunnskapen vi besitter. En fiktiv kursdeltaker sa hvor viktig det var for han/henne. En fiktiv sjef for en deltaker sa hvor viktig det var for han/henne.

Alt dette er jo fiktive uttalelser, men det det dreier seg om er hvordan suksess vil se ut.

Dette er da med på finne en visjon..?

Ja, det var da visjonen for det lille prosjektet, og denne var utviklet som en eneste stor pressemelding. Det er metode som forøvrig som vil beskrives i Mike Cohns nye bok, som kommer ut etter sommeren. Dette (pressemelding fra fremtiden) er bare en variant av en

god gammel ledelsesteknikk; Tenk deg at du har en tidsmaskin og at du går inn i fremtiden. Prosjektet var en braksuksess. Hvorfor det? Tried and true. Det funker.

Visjonen, er det vi anbefaler å bruke på alle Scrum-prosjekter. Det andre vi anbefaler at alle Scrum-prosjekter innfører er sprint-feiring, som et ekstra møte. De sprint-feiringene som jeg har vært innvolvert i selv, to stykker, har hatt kaffe og kaker samt visning av bilder fra sprinten. Det har også vært lunsj på en restaurant. Det behøver ikke å være dramatisk og man må ikke slå på stortrommen, men en markering.

Finner sprint-feiringen sted samme dag som sprint demoen da?

Ja

Hvor lenge er det mellom hver sprint? Fra en sprint slutter til neste begynner...

Det er dagen etterpå sprint reviewen. Jeg har aldri opplevd noen som ikke har gjort det. En vanlig sak er å avslutte sprinten på fredag og begynne neste på mandag. Det gjør vi faktisk ikke. På det prosjektet her, tro jeg faktisk det hadde vært bedre om vi hadde gjort det, men jeg vil helst ikke krangle på det. På veldig kort sprinter så har jeg hørt om team som har review, retrospective, lunsj og så sprint planning. Hvis du har 1-ukers sprint så vil du ikke bruke to dager på å vente.

Vi har hatt et prosjekt, som jeg ikke har vært med på, hvor det ble kjørt 1-ukers sprinter og da tror jeg at de gjorde noe slik.

- *Er det noen grep (som du ikke har navnt til nå) dere gjør, som ikke er en del av Scrum, men som dere likvel må ha med på et Scrum-prosjekt for at det skal fungere?*

Ja. Uten automatisert testing så blir det ikke noen fremdrift i praksis. Dette er et viktig punkt. Dette er noe Scrum ikke sier noe om. Hvis du hører på folk som Jeff Sutherland, så sier han at alle vellykkede Scrum-prosjekter gjør det. Det er en uskreven regel.

Jeg ser en del Scrum prosjekter som ikke går så bra som de kunne ha gjort fordi de ikke fokuserer nok på verdien som skal leveres. Da tror jeg at det å dra inn folk som har brukerfokus, i Scrum, kan hjelpe veldig.

Hvilken rolle vil disse personene (som har brukerfokus) da ha? Vil de da være supplerende for teamet eller vil de være på scrum master-nivå. Hvordan passer de inn i rollene som er etablert for Scrum?

Vi har to eksempler hvor vi har brukervennlighetseksperter på prosjektet nå. De har en rådgivende funksjon på tvers av flere team. Jeg ville heller hatt dem på teamet, men vi har ikke hatt nok av dem (til å kunne gjennomføre det).

Hvordan blir disse ekspertene valgt ut? Er det kunden som plukker disse ut eller er dette deres ansvar?

På det prosjektet som jeg har vært minst innvolvert i så er det én person som, jeg tror, er leid inn av kunden som en ren ressurs. På det prosjektet jeg er med på nå, så har vi én som er en nøkkelperson når vi skal levere et tilbud. Da leverte vi fulloppsatte team, og da inkluderte vi brukervennlighet på koordineringsteamene. Vi har to operative team og så har vi et lite team som består av prosjektleder for begge, brukervennlighetsekspert, forretnings-arkitekt for begge, testansvarlig for begge. Det er vanlig, men ikke ofte sagt at man har en supportorganisasjon eller ledelsesorganisasjon utenfor Scrum team.

Vi har normale team på 8-9 personer hver. Vi har også fem sentrale ressurser som går på tvers av disse teamene. Dette er vår del av prosjektet. Parallelt så går det en annen del. Det er en underleverandør som har en parallell organisasjon og så har kunden selv en parallellorganisasjon, som er cirka tre ganger så stor på begge postene. Totalt sett, med variasjoner, så har du tre kopier av dette. To små og en stor. Leverandører og konsulent-selskap som er de to små og kunden selv som er den store. Totaltomfanget er såpass stort at personalet på kundens eget delsprosjekt her inkluderer mange konsulenter. De er da innleid på tidsbasis, mens her er det konsulentselskapene som har leverandøransvaret. Det er her bedriften (bedriften han jobber i nå) kommer inn.

Dette viser hva som skjer når du putter Scrum inn i et prosjekt som er i hundervis av millioner klassen. Da har du noen ting som er veldig Scrum-aktig og så har du hele konteksten rundt som inneholder veldig mange elementer. Det som er bra er at vi ikke faller inn i fellen hvor man benytter Scrum inni fossefallsmetoden, ihvertfall ikke riktig enda.

Når det funker bra, kan vi si at vi kan levere verdi langs med den relative kundeaksen også har visse andre folk andre akser som de kan levere verdi langs med, hvis vi klarer å frikoble oss såpass mye. Vi kommer også borti steder hvor vi toucher hverandre, men vi har jo ikke kommet dit enda. Det blir gøy når vi kommer dit. Vi har noen som jobber med saksbehandlingsløsninger, det er oss. Vi har noen som jobber med eksterne, som gjelder webeksponering og den type ting. Vi har også noen som jobber med batch-jobber som skal kjøres for å få dette systemet til å funke. Vi har en felles database som man jobber rundt, og har den som et nav oppi det hele. Dette er nesten hvordan det er. Det er et unntak og det er at både herfra (peker på saks-behandlingsdelen) og herfra (peker på den eksterne delen) så har vi forretningsregler. Dersom vi gjør dette riktig så kan vil det bli lite kommunikasjon mellom saksbehandlingsdelen og den eksterne delen, slik at mesteparten av kommunikasjonen går direkte fra disse delene og til databasen. Det som f.eks kan skje er at saksbehandlerdelen sier; kjør *den* forretningsregelen nå, og da bør databasen være oppdatert basert på *den* kjøringen. Da kan det tenkes at den eksterne delen bruker denne delen av databasen samtidig. Det som da skjer i integrasjonen mellom saksbehandler-løsningen og forretningsreglene er at dersom folk ikke er klare med det som har med forretningsreglene å gjøre, så funker fortsatt saksbehandlerløsningen, men den kan ikke ikke gjøre en endring i databasen enda. Da har de en kobling, men den koblingen er veldig tynn. Det er her kompleksiteten begynner.

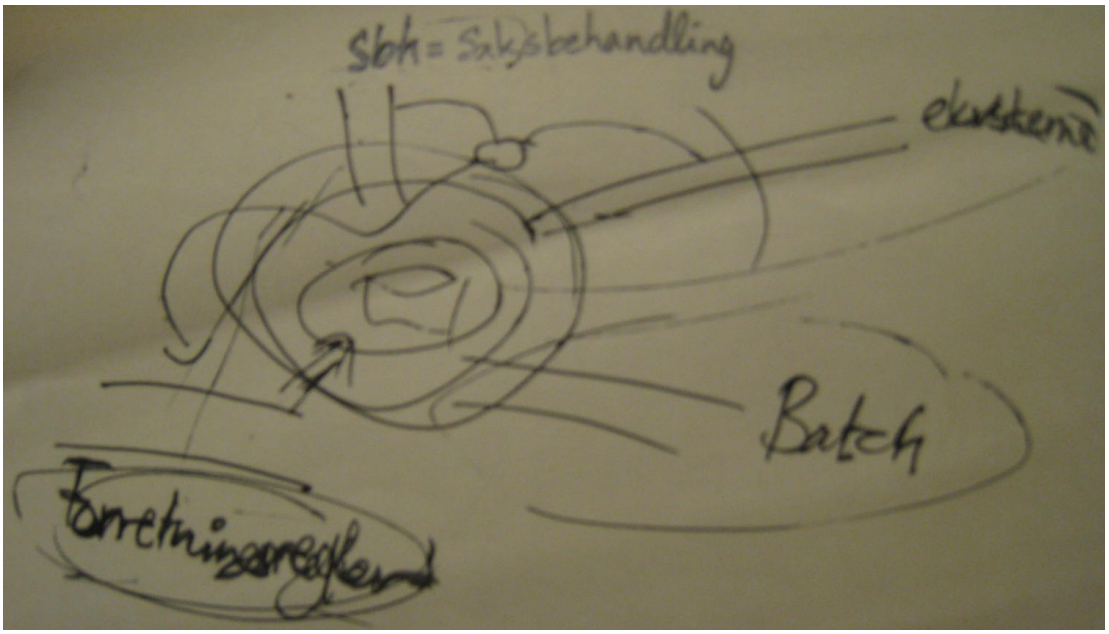

Figure B.6: Illustrasjon av saksbehandlingssystemet.

Det å separere ut de som skal jobbe mot eksterne og de som skal jobbe mot saksbehandlerne og de som skal jobbe mot batch-prosesser, er i utgangspunktet mulig å få til hvor et gitt team leverer noe som fungerer bra for sine brukere, mens en operasjon for andre brukere blir forsinket, eller omvendt.

Utfordringen er å angripe de stedene hvor de ulike delene av systemet ikke er frikoblet, men samtidig ikke vikle seg inn i unødig kompleksitet. En slik database som ligger i bunnen har en tendens til å akkumulere lag som perlemor rundt et sandkorn i en musling, bortsett fra at slike lag som akkumleres i applikasjoner som deles på tvers er sjelden så vakre som en perle. Det er masse feller å gå i, men det er også masse muligheter til å gjøre det bedre.

Har dere noen spesielle retningslinjer i forhold til når dere skal dokumentere og da evetuelt hvor mye som skal dokumenteres?

Da kommer vi inn på definition of done eller done-criteria. Et produktkøelement, det vil si en user story, deles opp i flere tasks. Før vi kan si oss ferdige med en task så skal vi gjort en del oppgaver. Vi har som en del av ferdigstillingskriteriet for user stories å oppdatere den felles dokumentasjonen. Det er en del av det som skal til for at en user story skal markeres som ferdig. Men jeg tror ikke vi er helt enige om hva nødvendig dokumentasjon er. Jeg tror aldri jeg har vært borti et prosjekt som ikke gjør det på den måten, hvor nødvendig dokumentasjon er en del av definition of done, så er kranglingen/uklarhetene egentlig; hva er nødvendig dokumentasjon?

- *Vi har ikke snakket så mye om planleggingsfasen. Hvordan fungerer planleggingen hos dere? Har dere for eksempel en pre-planleggingsfase?*

Det er en del folk som jobber med å få ferdig produktkøen til sprintplanleggingen sammen med *product owner* og forretningsanalytikere. Jeg vet ikke helt hva de gjør rett opp mot sprintplanleggingen. Vi har en parallell prosess som vi håper ikke skal ta forferdelig mye tid på sikt, men som forbereder til neste sprint og så til neste release.

I og med at dere har en prosess der som ikke er helt på plass enda, hvordan har dere gjort dette tidligere?

Vi har hatt en relativt stabil backlog som sjelden har endret seg mye fra sprint til sprint, så det har ikke vært noe pre-planleggingsmøte.

Er det product ownerne som velger ut elementer fra product backlogen og delegerer disse?

I den forrige bedriften jeg jobbet for hadde vi ikke *product owners* som var så bestemte. Det ble ofte en samtale rundt det (rundt product backlogen). I praksis så var det helst teamet som avgjorde hvor mange og hvilke elementer fra product backlogen som skulle være med i sprint backlogen.

Hvilken rolle i diskusjonen mellom product owner og teamet har scrum master? Hvordan stiller scrum master seg i forhold til valg av elementer til sprintbackloggen?

I den sammenheng faller *scrum master* ned som et team medlem (har samme status som et teammedlem). *Scrum master* pleier gjerne å være en som er litt mer høyrøstet.

I denne situasjonen i bedriften (den forrige bedriften han jobbet for) så hadde jeg en del påvirkning på hva som ble med (i sprint backlogen).

Hvordan gjøres dette, i den bedriften du jobber for nå, med tanke på den pre-planleggingsprosessen, som ikke er helt på plass enda?

Scrum master er ikke formelt sett med på pre-planlegging av sprinten. Formelt er det ingen fra teamet som er med på denne fasen, men man spør folk etterpå om det. Det som er planen for hva som skal skje, er at vi har en sprint hvor det er en leveranse som skal presenteres på demo-dagen. Dagen etterpå så planlegger vi og da skal det være en rimelig grovbygget product backlog som kommer fra *product owner* og forretnings-analytikere. De har nå jobbet parallelt med teamet under denne sprinten og så har vi hatt en del uformelle, men ikke noen formelle avsjekkingspunkter mellom nøkkel-personer på teamet, det være seg *scrum master* eller andre med kompetanse for å prøve å få dette på plass. Det har ikke vært noen avsjekkingspunkter hvor det har vært en del av prosessen. Dette er ikke helt som det kommer til å bli, men ting vil gå seg til etterhvert.

I bedriften (den forrige bedriften han jobbet for) så var product backlogene veldig stabile uten at vi hadde lagt for mye forarbeid i dem. På det prosjektet jeg er på nå (i den bedriften han jobber for nå), så er vi litt mer ustabile, men det er forsåvidt greit nok siden vi er såpass tidlig i prosessen.

Hvor lenge varer typisk et sprintplanleggingsmøte i den bedriften du jobber for nå?

Fire timer er jo standard. Vi sitter som regel fire timer sammen med kunden, men vi setter av hele dagen.

Pleier dere da også å bruke hele dagen til sprintplanlegging?

Vi bruker den i hvertfall ikke på så veldig mye annet. Som regel er folk så slitne på slutten (av møtet) uansett. Da har vi gjort planning poker og da har vi gjort diskusjon av hva som skal med og definert hva det er.

Hvordan, i hvilken grad og i hvilke situasjoner bruker dere planning poker?

Vi bruker planning poker på sprintplanleggingsmøtene. Vi bruker planning poker primært på stories, ikke så mye på sprint backlogen. Der er det litt uklar praksis. Det Mike Cohn, som står bak planning poker, sier er at han anbefaler planning poker på product backlogen, men ikke på sprint backlogen. Vi ser også at for sprint backlogen så liker folk bedre å planlegge ved bruke timer (til å estimere) uansett.

Hver gang det er blitt gjort en større endring av product backlogen, så bruker vi planning poker for å gå gjennom elementene som ligger på product backlogen. Da starter vi typisk på toppen så går vi gjennom product backlogen og bruker 2-3 timer til vi er for slitne til å fortsette og så kommer vi så langt vi kommer på product backlogen. Så blir det så bra som det blir.

Det som allerede ligger på product backlogen er user stories. I det nåværende projektet (i den bedriften han jobber for nå) nå og på det forrige prosjektet jeg var med på, så hadde vi user stories på formen; som .. så ønsker jeg at... Slik var det for alle elementene på product backlogen. Personlig så liker jeg å putte ikke-funksjonelle krav der også. For eksempel, ”som en utvikler ønsker jeg å bruke en raskere applikasjonsserver slik at jeg bruker mindre tid på å taste å verfisere endringene mine. Det som er interessant med den er jo det at, når vi formulerer den riktig så får vi *product ownerne* til å si; ”dere bruker litt mye tid, kan

dere ikke prioritere den oppgaven som går ut på at ting skal gå raskere." Både oppgaver som går på brukbarhet og ytelse i utviklingseffektivitet skriver vi gjerne på den formen. Det er det som ligger på product backlogen og det er det vi estimerer.

Vi kjører helst sprintplanleggingen i to deler. Den ene delen er *product owner* med på og da starter vi fra toppen av det vi tror er den prioriterte product backlogen og så kjører vi planning poker på alle oppgavene nedover. Underveis så kommer *product owner* med oppklaringer. Dersom *product owner* ikke kan oppklare (et problem) eller ikke er til stede, som også har skjedd, så skriver vi ned en hypotese og så sjekker vi den med *product owner* etterpå.

Så plukker vi ut de viktigste storiene fra product backlogen, shuffler de litt om hverandre slik at vi får noe vi kan starte med. Da kan *product owner* egentlig få lov til å forlate møtet. Da bryter vi de utvalgte storiene ned i tasks.

Dette er den prosessen vi har kjørt på de to teamene som er hos kunden på dette prosjektet (prosjektet for den bedriften han jobber i nå). I den forrige bedriften jeg jobbet for, så var det en langt mindre strukturert prosess. Da valgte vi faktisk å ikke estimere. Hvis du har en product backlog som *product owner* ikke er interessert i å endre på, så er det ikke noen vits i å estimere uansett. Hvis du skal ha det uansett hva det koster, så la det koste det det koster.

B.5 Interview with *scrum master* in project-13, 29.April 09

❖ *Hvor mange er dere i hvert team?*

Nå kan jeg kanskje ta litt om bakgrunnen for hva jeg har jobbet med i Scrum. Jeg har jobbet i et prosjekt med en kunde der vi skulle lage et veiinformatikksystem. De hadde et gammelt system og vi skulle ordne et nytt system pluss at de skulle ta over to andre systemer, så det var en stor migreringsjobb som måtte gjøres. Jeg startet vel høsten 2006, første november. Etter å ha vært et halvt år som prosjektleder i dette prosjektet, altså på våren 2007, i mai, så gikk jeg på *scrum master* kurs. Da så jeg at en del av de tingene som Scrum hadde kunne passe inn i prosjektet. Dette er et prosjekt ute hos en kunde, hvor jeg var eneste deltaker fra bedriften jeg er ansatt i. Vi (i bedriften) brukte ikke Scrum i noen andre prosjekter. Det jeg gjorde var å innføre elementer fra Scrum i prosjektet. Det første

vi gjorde var å ta daglig møte (daily stand-up), *product backlog*, *sprint backlog*, slike ting som var enkle å innføre. Vi definerte sprinter selv om vi hadde en leveranseplan og ting som skulle gjøres var fast definert. Vi kunne ikke kjøre 4-ukers leveranser, vi kjørte 6-ukers leveranser stort sett. I tillegg hadde vi ansvar for det gamle systemet. Det vil si vi måtte drive support samtidig som vi skulle drive nyutvikling av den nye funksjonaliteten som skulle komme i det nye systemet. Det var litt gjenbruk og litt nyutvikling. Det var settingen for det prosjektet som jeg var i.

Vi var rundt 7-8 utviklere i teamet. Alle sammen var spesialister innenfor sitt område. To av dem var arkitekter, så du kan si de hadde kompetanse på kjernedelen. To stykker jobbet med økonomi-delen, og to stykker jobbet med presentasjonsdelev, dvs. brukergrensesnittet. Dermed var det aldri mer enn to personer som kunne overlappe hverandre. Men som regel var de spesialister innenfor hvert området igjen, så det var ikke slik som det vanligvis er i et vanlig Scrum team hvor du bare kan gå å ta en oppgave. Grunnen til det var at alle var avhengig av hverandre, at alle gjorde sine oppgaver for at det skulle være mulig å sette sammen et system. Vi hadde med teknisk prosjektleder og en som hadde ansvar for infrastruktur som vi i tilllegg hadde med i de daglige møtene (daily stand-up). Vi var ganske mange til stedet i disse møtene, men vi klarte å kjøre dem på et kvarter likevel.

Hvor lenge har du jobbet med Scrum for bedriften du er ansatt i nå?

Jeg begynte i mai i 2007 og siden så har jeg jobbet med det frem til nyttår (2008, dvs rundt 1.5år). Da var jeg ferdig i det prosjektet (prosjekt-13). Da brukte vi ikke alle elementene av Scrum. Vi hadde for eksempel ikke *product owner*. Det var litt spesielt for kunden kunne mindre om systemet enn utviklerne, så vi måtte bestemme rekkefølgen (på kravene i *product backlog*'en) selv. Vi hadde et system som var detaljert kravspesifisert fra før (når det gjaldt) hva som skulle lages. Det var en logisk rekkefølge vi måtte levere dem ut i, så det var ikke det store behovet (for en *product owner*). Vi hadde en stor product backlog som ga mer enn nok å jobbe med i lang tid fremover, så vi gjorde prioteringer selv i forhold til behovet hos brukerne i product backlogen; hva vi skulle ta inn i neste sprint.

- *Hvor lenge har hvert team jobbet sammen? Hva er typisk her og hvilke fordeler/ ulemper er det i forbindelse med utskiftning av team medlemmer?*

Nei, prosjektet gikk hele tiden, men vi utvidet med folk etterhvert. Det kom inn..

Ble flere team dannet da?

Nei, noen sluttet også, så vi hadde vel mer eller mindre konstant antall personer (på teamet). Vi startet med syv (team medlemmer) og så var vi vel oppe i ni på det meste. Men det var i korte perioder. Det var to sentrale ressurser som sluttet i prosjektet, mens vi kjørte så det var stort sett konstant antall utviklere. Det kom nye ansatte inn som både skulle lære seg teknologien og lære seg produktet. Det var ganske utfordrende.

❖ *Hvordan løste dere det? Hadde dere kursing da eller andre tiltak?*

Nei, for Scrum så kjørte vi bare en kort-versjon av hva vi driver på med og så fikk de bare følge med. Vi kjørte altså en introduksjon av Scrum for teamet.

❖ Var det for eksempel *scrum masterne* som tok denne jobben?

Jeg tok jobben som *scrum master*. Jeg var prosjektleder før vi begynte med Scrum og så kjørte vi en evaluering etter de første rundene (sprintene) om vi skulle fortsette og ta inn flere elementer av Scrum. Folk var interessert og syns at det hadde fungert bra i den første sprinten og da kunne vi jobbe videre med det og ta inn nye elementer. Det var ikke ***ekte*** Scrum slik som vi ville ha kjørt det i et Scrum team der du disponerer alle ressursene og klarer å skjerme dem fra alle forstyrrelser. Det som det endte med på slutten var at vi ikke klarte å gjennomføre den siste sprinten, for vi ble rammet av vedlikehold, konverterings-problemer som bare dukket opp. Du må bare stoppe utviklingen og gjøre andre ting. Det er slike ting du oppdager i det praktiske liv, dersom du har support og feilretting på gamle system i tillegg til utvikling så stopper det. Dessuten var det én sentral ressurs som kunne det gamle systemet best og retting i det, som også var sentral i det nye systemet, så når han ikke fikk utviklet sin del av funksjonaliteten så ble de andre litt hemmet av det igjen. Det var også en ting vi prøvde å jobbe med – å få flere folk slik at man har større dekning innenfor samme kunnskapsområde.

❖ *Kan du fortelle litt om hvordan deres Scrum of Scrums møter foregår, hvem som deltar og hva som typisk blir tatt opp på slike møter? Hvor ofte foregår slike møter?*

Det gjelder helst et prosjekt vi kjører i Oslo som går for en annen kunde (dette er prosjekt-15). Der kjører vi Scrum of Scrums. Jeg har ikke vært i dette prosjektet. Jeg vet at de har en rutine på å kjøre Scrum-møter (daily stand-up) for de 5-6 teamene og så rett etterpå

samles *scrum masterne* og tar en Scrum of Scrums. Da kjører de først et kvarter (hvor daily stand-up møtet avholdes) og neste kvarteret så kjører de Scrum of Scrums der de treffes for å løse problemer.

Én eller to ganger i uken så kjører de Meta Scrums. I disse møtene tar de inn andre, altså folk som egentlig ikke tilhører Scrum team. Dette er driftsoperatører og folk som sitter i produksjon eller på en annen måte utenfor prosjektet, slik at de kan få løst opp i eksterne problemer. Prosjektet er avhengig av at disse personene kommuniserer og at de får ting på plass.

Er det vanlig at høyere ledelse er med på Meta Scrums?

Prosjektleder i dette prosjektet var ikke med i det hele tatt. Det var ikke ledelsen som var med. Det var mer fagfolk innenfor forskjellige områder som satt på møtene.

Du snakket også om at dere brukte 6-ukers sprint lengde..

I det prosjektet som jeg var i så var det seks uker, men på prosjekt-15 så brukte de fire uker. Erfaringer fra prosjekter som vi har hatt er at dersom du har flere parallelle team så er det fast lengde (på sprintene) og alle har samme syklus. Da får du en rytme i prosjektet, slik at alle vet nårtid ting skjer. Vi kjørte stort sett 6-ukers sprinter, men det var fordi det var planlagt tid til å få ferdig funksjonalitet. Det var funksjonalitetsstyrt det vi gjorde. Vi kunne ikke bruke mindre enn seks uker før alle var ferdige med sine oppgaver. Det var det som var årsaken til at vi kjørte 6-ukers sprinter. Dersom vi hadde hatt større team så kunne vi ha kjørt raskere, men utfra de ressursene vi hadde så klarte vi å produsere oppgaver som var leverbare etter seks uker. Det var enkelte som kanskje ikke hadde mer enn to dager i uken tilgjengelig til å produsere (dvs. skrive kode i forbindelse med nyutvikling). Ellers jobbet de mest med vedlikehold av det gamle systemet, feilretting i det nye (systemet) og kundehenvendelser. Dette spiste av utviklingstiden. Vi kjørte kun product backlogen mot utviklingen av det nye systemet, men support og slike ting ble det bare sagt at skulle være på siden, og så reduserer vi heller tilgjengelig tid de har for å produsere. Det kan være vanskelig å forutsi henvendelser. Vi måtte bare prøve å begrense tiden som ble brukt på support, for det kom masse henvendelser - nye brukere av systemet som ikke skjønner alt og man får mange dumme spørsmål og da er det en som trekker vekk de dumme henvendelsene, men så må det jo svares på en del ting da; ”Hvorfor fungerer ikke dette?” Så viser det seg at det fungerer, men på en litt annen måte enn det de var vant med fra før.

Dersom det kommer en oppgradering av et system så er det alltid noen som ikke skjønner hvordan det nye fungerer. Da blir det support på slike saker også. Så blir det jo også oppdaget feil, det er jo et veldig komplekst system, som også må rettes og det blir jo prioritert. De alvorlige feilene. Så skal det introduseres nye brukere hele tiden som skal inn i systemet, konvertering av data og opplæring. Det er masse eksterne aktiviteter som du ikke har kontroll over - ikke alle hvertfall.

Når det gjelder sprintplanleggingsmøter i forkant av hver 6-ukers sprint, kan du fortelle litt mer detaljert om hvordan disse foregikk?

Vi hadde jo en liste med ting som skulle lages, altså en produkt backlog som vi gikk gjennom og plukket ut ting som vi måtte lage til neste leveranse. Vi var på etterskudd hele tiden, så det sa seg selv (hva vi skulle prioritere etter) hva kundene hylte mest etter å få. Vi gikk da igjennom listen estimerte og gjorde da detaljestimater på arbeidet for hver enkelt person, for deres oppgaver. Det ble satt inn i et sprint-regneark. Det vi gjorde var at vi brukte samme regnearket og satte sprintnummer på de ulike oppgavene slik at vi kunne filtrere på hvilke oppgaver som gjaldt den (nåværende) sprinten. Vi hadde også folk som jobbet med aktiviteter som ikke gjaldt den nåværende sprinten, men som ville komme i neste sprint. Dette var fordi de hadde ledige kapasitet til å gjøre det. Dette var heller ikke helt Scrum-aktig. For eksempel folk som jobbet med rapporter - en rapport som vi visste at vi ikke trengte i inneværende sprint, men ville komme i senere, dersom en hadde alt for lite å gjøre i denne sprinten, så jobbet man med oppgaver som kom senere. Vi tok det ikke med i sprinten fordi det vil ikke være en leveranse i den sprinten. La oss si at vi var i sprint fire, da jobbet de aller fleste på oppgaver for sprint fire, men da hadde han (en av utviklerne) allerede begynt på oppgaver som var definert for sprinten etterpå. Vi brukte det (nevnte) regnearket, estimerte i timer og prøvde å bryte det ned til maks to dagsverk. Så kjørte vi burndown (chart) i det samme regnearket. Vi hadde et regneark som hadde både product backlog, sprint backlog og burndown (chart).

Det første jeg tenker på, siden dere egentlig ikke hadde product owner på dette prosjektet, når dere da kjører sprint planning møter og plukker fra product backlog, hvordan foregår da kommunikasjonen med kunde? Er det noen representanter fra kunden til stedet når dere har sprint planning møter?

Ikke på det (møtet). Det er litt spesielt dette systemet, fordi kunden var veidirektoratet og veidirektoratet hadde laget kravspesifikasjonen, men de visste ikke hvordan systemet skulle brukes etterpå. Det vil si det var andre, driftsoperatør-selskap som skulle bruke det, og vi visste hva de ville ha. De kunne påvirke kravspesifikasjonen, men kunden som sådan hadde bare brukbar kjennskap til det, men ikke veldig god kunnskap. Når vi skulle definere hva som skulle være med i neste sprint, så var det basert på innspill som var kommet; "Nå må vi ha dette her på plass." Vi snakket med driftsoperatører (brukere), vi snakket med veidirektoratet (kunden), og så var det jo en prosess i bunnen av dette systemet som gikk på at du først får sørget for at de delene som samler inn data, så skal det ligge der for å bli avregnet mot kontoer. Først kommer jo uansett abonnementsystemet. Du skal avregne dette og så skal du se når betalingen kommer inn og så skal du sende en purring, og så er det innkasso. Avregning – purring – innkasso. Dette var jo et system som de hadde fra før, og vi skulle bygge videre på det. Altså det var videreutvikling av et gammelt system, og da ga det seg selv hvilken funksjonalitet som du måtte ta med i neste leveranse, altså hva du måtte ha ferdig til neste leveranse. Det (funksjonaliteten) kom først inn når de (som skulle begynne å bruke systemet) skulle til å bruke den. Så tre måneder etter at det (systemet) var blitt tatt i bruk så skulle de begynne å sende ut de første inkassovarslene, og da fikk de inkassomodulen. Det var bare såvidt de fikk den (inkassomodulen) til de trengte den. Prosessen var slik at du hele tiden oppgraderte funksjonalitet. Han som var arkitekten for det gamle systemet, visste hva som måtte gjøres, så det ga seg i stor grad selv hva som måtte være med (i hver sprint backlog). Men i tillegg så var det jo ny funksjonalitet som vi prioriterte rekkefølgen på sammen med veidirektoratet (kunden). Vi hadde en liste som var prioritert, så tok vi bare fra denne listen og gjorde de valgte tingene først. For eksempel automatisk, dvs. optisk lesing av bilnummer på bilder, det laget vi ganske tidlig for å gjøre det manuelt (når folk skal jobbe med å avlese disse skiltene selv) er ganske omfattende. Det er mye arbeid, så ting som kan automatiseres tok vi først. Du visste godt hva du måtte lage så lenge du hadde hele systemet definert (på forhånd). Vi hadde ikke behov for *product owner* i det systemet. I ettertid vil de ha bruk for det, for da vil de få nye krav for da skal de levere (systemer) for andre. Da skal de levere i Portugal og Slovakia og slike

steder og da vil det komme andre krav inn i systemet. Dersom vi skal prøve å beholde et system med minst mulig varianter av systemet, blir det mer aktuelt å tenke på i hvilken rekkefølge skal du gjøre det og hvordan skal du gjøre det. Da kan du ha en *product owner* der som definerer at *det* kommer før *det* osv. (prioriterer krav). Da trenger du den funksjonen (*product owner*-funksjonen), men så lenge vi hadde et system og stort sett skulle gjøre forbedringer og tillegg til det gamle systemet så ga det (prioriteringen av kravene) seg selv. Men også fordi at den største eksperten på systemet var han som var arkitekten for det gamle. Han visste hva som skulle til for at det systemet skulle gjøre jobben i tide.

- *Med din rolle som scrum master i dette prosjektet, var det noen spesielle tiltak du måtte gjøre eller noen oppgaver du følte var viktige å få til som scrum master i dette prosjektet?*

Jeg startet som prosjektleder, men så hadde jeg lyst til å innføre Scrum og gjorde det som et forsøk og var da *scrum master*. Ut av prosjektet, mot ledelsen, til kunden og brukerne så var jeg prosjektleder. Så *scrum master* var bare inn mot utviklerne. Da jeg sluttet i prosjektet så ble det omorganisert, slik at vedkommende som var teknisk prosjektleder som hadde mer den interne kommunikasjonen ble *scrum master*, mens prosjektleder ble prosjektleder. Vi splittet den rollen. I begynnelsen så var det bare jeg som visste hva Scrum var. Da gikk teknisk prosjektleder på kurs og lærte seg det, og hadde vært med på så mange møter og diskutert prosessen slik at han kunne overta som *scrum master* da jeg sluttet og da kom det inn en ny prosjektleder. Prosjektlederen har i hovedsak kontakt ut av prosjektet, mens *scrum master*/teknisk prosjektleder har det interne ansvaret. Det var jo innføring av Scrum i et stort komplekst prosjekt som allerede var i full gang og man måtte gjøre det forsiktig, ikke gjøre for mange endringer på en gang.

Var det noen spesielle utfordinger, du fikk som scrum master, etterhvert som prosjektet skred fremover og dere ble bedre kjent med Scrum, da tenker jeg både på utviklere og andre folk som var involvert?

En ting var det at folk ikke var så veldig flinke til å oppdatere estimert gjenstående hver dag (dvs. å oppdatere et felles tilgjengelig excel-regneark slik at det reflekterte hvor mye som gjenstod av oppgaven de jobbet med. Dette var viktig i forbindelse med å vedlikholde burndown chartet, som også lå i dette excel-regnearket). I starten så fikk vi gjort dette én

gang i uken, men vi ble enige om at vi skulle kjøre det (oppdatere regnearket) hver dag og fikk aksept for det, men det var mange som oppdaterte, estimert gjenstående på sine oppgaver, kun hver andre eller tredje dag. Det var den ene utfordringen.

Den andre gikk på at teamet skal være beskyttet mot eksterne henvendelser og bli motivert til å nå de målene de har satt seg. Det fungerte ikke fordi de ble hele tiden bombardert med henvendelser fra kunder, feilretting og andre ting som også måtte prioriteres. Det gamle systemet måtte fikses og de fikk henvendelser angående dette. De måtte drive med salgsstøtte i forbindelse med salg av løsning til nye prosjekter. Du setter deg et mål, og så blir du hindret i å nå målet, på grunn av alle disse eksterne tingene (henvendelsene). Det gjorde at teamet aldri klarte å få den motivasjonen du får ved å nå målet. Slik man i teorien skal få med Scrum, hvor 1 + 1 blir 3. Det var en utfordring. Det var masse krangler med ledelsen for å få minst mulig forstyrrelser, eksempelvis at folk ble kalt inn til møter eller måtte gjøre andre typer oppgaver. (Vi prøvde å få det til slik at) alt gikk via meg, men det fungerte ikke alltid. Det er vel også ulempen med så lange sprinter, at de ikke kunne vente, de måtte ta det inn midt i en sprint. Hadde det vært kortere sprinter, så kunne vi sagt; ok, vi er ferdig om 3-4 dager, da kan dere få låne ressursene en stund.

Ellers var det ingen av Scrum-tingene så var noe utfordrende. I begynnelsen brukte vi kanskje daily scrum møtene (daily stand-up møtene) mer til statusrapportering og ikke så mye på fremdrift, men vi kjørte fremdrift gjennom burndown chart som hver enkelt oppdaterte. Vi tok så en ny utskrift (av burndown chartet) og hengte opp på veggen hver dag. På den måten kunne alle se at kurven gikk nedover. Vi hadde ikke et opplegg hvor man har oppgavene stående på et (task) board. Vi brukte regnarket istedet som et felles medium. I ettertid så kanskje vi kunne tjent litt visuelt på å hatt et (task) board der man kunne se at man flyttet hovedoppgavene hvertfall. Alle skulle oppdatere regnarket hver dag med estimert gjenstående.

Vi hadde også daily scrum (daily stand-up meeting) der vi fortalte hva vi hadde gjort osv. Det var det første vi startet med. Det er vel det mest effektive med hele metodikken (Scrum-metodikken), akkurat denne daily scrumen (daily stand-up møtet); ”Hva har vi gjort?”, ”Hva skal vi gjøre?” og ”Hva slags problemer har vi, hva er det hindrer oss?” Dette kjørte vi konsekvent. Men det vi ikke gjorde var; ”Hvor mange timer har vi gjenstående?” Det tok vi ikke på møtet. Det gjorde hver enkelt i regnearket, innen en gitt frist. Det var ikke slik at vi sto og noterte; ”i går hadde jeg åtte timer gjenstående og i dag

har jeg åtte timer gjenstående." Folk måtte ikke forklare i detalj hvorfor de ikke hadde kommet videre. Det er jo fordeler og ulemper med det (dersom de hadde gjort det så detaljstyrt). Folk blir jo på en måte presset mer, men på en annen side så må du begynne å forklare hvorfor du ikke har fått gjort det du planla at du skulle gjøre i går. Med alle de eksterne hindringene vi hadde, så hadde vi doblet tiden på møtet (dersom dette skulle tas opp i slik detalj på møtet). All (i teamet) vet jo grunnen til forsinkelsene (i og med at de eksterne henvendelsene var velkjente).

Gjorde dere noen spesielle grep, som vi ikke har vært innom, for å tilpasse Scrum til akkurat dette prosjektet eller til organisasjonen som sådan? Gjorde dere noen modifikasjoner av Scrum, utenom at dere ikke tok med alle Scrum-elementene? Var noen elementer dere la til for eksempel, som ikke er typiske for Scrum?

Nei, jeg tror ikke det. Vi hadde planleggingen i Microsoft Project før vi begynte med Scrum, men det kuttet vi ut etterhvert. Vi kjørte kun estimert gjenstående, og så kjørte team-oppfølging i tillegg; "Hvor mange timer har vi brukt på prosjektet?" I løpet av en sprint-periode så kan du si at vi hadde 500 timer tilgjengelig til produksjon eller det vil si oppgaver var estimert til ca. 500 timer med sprint. Deretter så vi hvor mange timer folk har ført totalt. Da så vi at folk hadde brukt omtrent dobbelt så mange timer på de oppgavene som folk har hatt ansvar for i løpet av sprinten. Da så vi at estimatene våre var feil. I neste sprint så satt vi opp at vi hadde en produksjon som var adskillig dårlige enn den vi planla med og da må vi finne ut hvorfor. Men det var jo som regel det at folk ikke klarte å jobbe effektivt eller at folk ble forstyrret og så førte de ikke alle timene på forstyrrelsene. Det var jo også feil estimater, men det går på estimeringsprosessen også. Hvor god var den? Vi la på ting når vi så at f.eks forrige gang så brukte vi dobbelt så mange timer på å produsere 500 timer, og vi rakk ikke mål likevel. Neste gang så må vi ta høyde for det at teamet ikke produserte i forhold til estimatene våre, så vi må da legge på marginer i forhold til det.

Jeg kan se for meg at å utarbeide team velocity kan være veldig vanskelig siden det er så mange (og så varierende mengde) avbrytelser?

Jo, men det blir jo på en måte det (denne team-oppfølgingen) som blir (team) velocity. Vi hadde jo egentlig timer som måltall, f.eks de oppgavene her er estimert til 500 timer. Deretter så vi at vi brukte 1000 timer til å bli ferdige med oppgavene som vi trodde skulle ta 1000 timer, så vi får et visst anslag på (team) velocity, selv om vi kanskje ikke bruker

funksjonspoeng. Vi brukte timer (til å estimere oppgavene) og da brukte vi timer til oppfølgingen også. Der hadde vi et eget system for å følge opp antall timer som ble lagt ned i prosjektet hver uke. Da førte vi det bare opp; så og så mange timer ble jobbet på sprinten denne uken. Så fulgte vi det opp fra sprint til sprint. Da fikk vi jo en idé om (team) velocity. Vi så også på hvor mange oppgaver vi hadde igjen da perioden (sprinten) var over. Vi så altså på hvor mange oppgaver vi hadde klart å produsere. Hvis vi (f.eks) skulle produsere 500 oppgaver, så hadde vi kanskje bare klart å produsere 90% av de oppgavene. Vi hadde grunnlaget for å se på (team) velocity og i starten brukte vi av og til dobbelt så lang tid som vi hadde estimert. Dermed hendte det at vi måtte kjøre en sprint én uke lenger. Det har jeg vel ikke nevnt. Sprintene var seks uker, men vi var avhengig av å levere funksjonalitet. Så dersom vi ikke var ferdig med funksjonaliteten, så fikk vi bare utsette leveransen én uke. Så vi hadde ikke konstant (sprintlengde). Vi måtte levere inn den (spesifikke) funksjonaliteten og da avsluttet vi sprinten da vi var ferdig med den planlagte funksjonaliteten, ikke når tiden var gått ut. Det er jo et avvik i fra Scrum. Men når du har team med én eller to personer som kan gjøre en oppgave i et såpass komplekst prosjekt (så blir det vanskelig å fullføre sprinten innenfor fristen). Normalt sett burde du sikkert hatt et dobbelt så stort team for å ha gjort dette (prosjektet) eller kanskje gjort det med tre Scrum team eller noe slikt. Ett team på kjernesystemet, ett på økonomi og ett på brukergrensesnitt. Vi har kanskje hatt nok på brukergrensesnittet, men på de andre så var det (for lite). Det var et underbemannet prosjekt i tillegg. Det er bare én som kan det (som har kunnskap innenfor hvert område) og når han ikke får tid til å gjøre jobben sin så blir man ikke ferdig. Men sånn er livet. Han som satt med brukergrensesnittet var jo avhengig av at han som satt med kjernesystemet hadde laget alle grensesnittene ferdig før han kunne få gjort sin jobb. Men det var lagt opp til nok oppgaver slik at det ikke ble ledig kan du si. Det var ikke noe ledig tid. Det var mer enn nok oppgaver som hver enkelt kunne ta ansvar for. Det var greit sånnsett å være på etterskudd. Da har du ting som skulle vært gjort. Dette prosjektet var ikke ideelt for å kjøre scrum. Men det med å ha leveranseperioder, daglige møter, sprintplanlegging og retrospektiv, er så positivt at det i seg selv gjør at du forbedrer deg. Daglige oppdateringer av status. Du får tenkt hva var det som funket og hva var det som ikke funket. De elementene er så positive at det lønner seg å ta de inn i prosjektet.

- *Kan vi snakke litt mer om hvordan dere utførte retrospektivmøtene?*

Vi samlet hele teamet og gikk gjennom; ”Hva var bra?”, ”Hva var dårlig?” og ”Hva kan vi gjøre bedre?” Da var det teamet og *scrum master* som deltok.

- *Hvor lenge varte typisk et retrospektivmøte?*

Vi hadde det så travelt at vi kjørte det på 1-2 timer etter hver sprint. Det er klart at vi gjorde jo endringer underveis i prosessen. Det ble laget nye verktøy enn dette regnearket, slik at arbeidsoppgavene til folk kom inn i en slags form for product backlog. De hadde jobblister; disse oppgavene er mine som jeg skal jobbe med fremover. Det var ikke bare prosjektet. Det ble litt mer struktur over input’en til hver enkelt - hva de hadde å gjøre. For prosjektet som sådan så var det en ulempe, men for hver enkelt person og for firmaet så var det en fordel fordi du fikk mer struktur på arbeidsoppgavene deres. Vi gjorde en del tilpasninger i forhold til det, men det har jo ikke noe med Scrum å gjøre det da. Det ble ganske komplekst etter hvert, ikke sant. ”Hva skal jeg prioritere?” Skal jeg prioritere support eller denne funksjonaliteten, for den må jeg bli ferdig med, og så må jeg på et møte for en selger skal ha en presentasjon. For meg var det én ting som var viktigst, og det var prosjektet. Spesielt var det kanskje supporten av det gamle systemet som (skapte mest problemer ved at ressurser ble ”stjålet”). Det kom folk fra Hellas skulle ha presentasjon. De (team medlemmer) hadde levert et system der for mange år siden og så skulle de overta, så skulle de sette av tid til det. Så kommer alt i prosjektet på etterskudd. Det var en utfordring. Vi var et ganske stort team som møttes. Vi var rundt 10 stykker. Nå var det jo ikke alle som var på disse retrospektivmøtene, men han som var infrastrukturansvarlig han var ikke på dette (på retrospektivmøtet). Han var til stedet på daily sprint (daily stand-up møtet) bare fordi han hadde noen små oppdrag. Han kunne være med å løse oppgaver som dukket opp underveis. Vi var vel rundt 10 stykker som var på hvert retrospektivmøte. Vi har ikke lyst til å bruke så mange timer på det. Du kan si du prøver å begrense tiden du bruker på overhead. Folk hadde jo andre ting de skulle gjøre.

- *Når det gjelder input fra kunden på ting dere allerede har produsert i løpet av en sprint, da tenker jeg typisk på review-møter hvor funksjonalitet blir demonstrert for kunden, hvordan løste dere det?*

Vi hadde akseptansetesting av systemet i henhold til kravspesifikasjonen. Vi laget først funksjonalitet og så kjørte vi akseptansetest på det. Når vi var ferdig med de to store

akseptansetestene, så hadde vi en liste med ting som ble kjørt gjennom, men vi hadde aldri en godkjenning fra kunden sin side. Vi bare leverte det rett til produksjon når vi var ferdig med det. Så vi kjørte ikke noen demonstrasjon. Det gjorde vi ikke. Én gang gjorde vi det. Det var først gangen (etter første sprinten), ellers gjorde vi det ikke. Da ble det bare satte rett ut i produksjon. Men det var heller ikke ordentlig Scrum i forhold til det. Det var også et litt spesielt forhold for kunden var ikke bruker av systemet. I tillegg til dette systemet så skulle det flyttes bompengeanlegg med egne databaser fra det gamle til det nye systemet, så det ble konvertering, opplæring og godkjenning, fordi hvert anlegg måtte settes opp på sin spesielle måte. Det ble kjørt akseptansetester pr. anlegg og vi hadde vel 15-16 anlegg. Da godkjente vi ikke hovedfunksjonaliteten til systemet. Da godkjente vi leie, at faktura og mer slike administrative ting som logo o.l. mer en personalisering av systemet. Det var det, det ble kjørt akseptanse (akseptansetester) på. Det ble gjort på et par timer på hvert anlegg, så det var vi utrolig effektive på.

Når det gjelder akseptansetestene, det som skulle testes og hvordan det skulle testes, ble dette satt opp under sprint planleggingsmøte?

Nei, det var satt opp på forhånd før vi begynte med Scrum. Det var veldig generelle krav i som var satt opp i forhold til det eksisterende systemet. Vi hadde en kravspesifikasjon som hadde rundt 700 krav som skulle tilfredstilles. Da dokumenterte vi alle krav etterhvert som de ble tilfredstilt og så hadde vi resten utestående, så tok vi en akseptansetest nr. 2 som tok resten av de resterende kravene. Så leverte vi det bare til produksjon uten at noen egentlig testet det. Fordelen var jo at det fantes et eksisterende system, slik at funksjonaliteten skulle være veldig lik på det meste.

- *Når det gjelder for eksempel dokumentasjon, under utviklingsprosessen, hadde dere noen spesielle retningslinjer dere fulgte i forhold til hvor mye eller hvordan dere skulle dokumentere?*

Ikke noe spesielt som vi tok ut av Scrum. Vi kjørte samme krav til dokumentasjon i forhold til design, i forbindelse med økonomi-delen var det krav om at et revisjonsfirma (skulle gjennomgå dette). Det var nokså strenge krav til dokumentasjon på de fleste områder. I fra design-nivå til beskrivelser av hvordan ting hang sammen og testing av alt. Vi hadde eksterne revisorer som var inne å sjekket at regnskapssytemet eller økonomisystemet var i henhold til lover og regler. Det var ganske omfattende kontroller.

Scrum ga oss ikke noe dokumentasjon annet enn det du slapp gjennom å ha de daglige møtene (daily stand-up møtene). Du slapp den interne kommunikasjonen i form av statusrapporter og møtereferater eller den typen ting. Det slapp du jo. Men ellers var dokumentasjonen på samme nivå som den var uten Scrum.

Brukte dere noen spesielle teknikker fra XP, som for eksempel par programmering eller lignende underveis..? Det er gjerne litt vanskelig siden dere hadde så veldig spesialiserte roller, men hendte det for eksempel at de som jobbet på GUI par programmerte?

Utviklerne satt i et stort rom, (hvor) to og to (satt) i mot hverandre. De to som jobbet på f.eks brukergrensesnittet satt i mot hverandre. Det var ikke par programmering, men det var en veldig direkte dialog mellom dem. Dersom den ferskeste hadde noe han lurte på så hadde han muligheten til å spør med en gang face-to-face. Så plassering av utviklerne var veldig bra, i forhold til intern kommunikajson. Det var veldig fritt for diskusjoner og møter internt på hvordan man gjør ting og hvordan løses det her. Noen lager kjerne og noen lager brukergrense så tar de en diskusjon på det. Men vi hadde ikke par programmering som sådan. Det tror jeg ikke ble brukt.

- *Brukte dere noen andre teknikker om er kjent fra XP, som test-drevet utvikling eller continous integration?*

Nei. I og med at dette var et gammelt system så ble portet over til en ny platform og slike ting. Dersom vi hadde startet fra scratch så kunne man bygd det inn, men å starte i et gammelt system og gjøre det der er veldig krevende. Det er sannsynligvis ikke effektivt heller. Jeg leste en artikkel på det til og med. Det var noen som har erfaring med det.

Jeg sluttet i prosjektet ved nyttår (2009). De satt inn en intern prosjektleder. Jeg var innleid som prosjektleder. Prosjektet pågår enda. Det er et prosjekt som går over mange år. Oslo skal inn i systemet, bare å få flyttet Oslo inn i det nye systemet er jo et prosjekt i seg selv, og de skal ha masse tilleggsfunksjonalitet. Det er mer et forvaltningsprosjekt, enn et nyutviklingsprosjekt. Videreutvikling og vedlikehold av et gammelt system. Så det foregår fremdeles.

- *Etter at dere hadde kjørt prosjektet etter en stund, hvor smidige følte du at dere var? Dersom det kom inn uventede endringer, hvordan klarte dere da å tilpasse dere det?*

Det var ingen uventede endringer i forhold til funksjonalitet. Vi var fortsatt på et nivå der ting måtte gjøres. Vi hadde en liste over ting som måtte gjøres, så det var bare å prioritere de tingene som var definert. Vi hadde jo noen utfordringer underveis, for eksempel den bildebehandlingen (bilder tatt av bilskilt). Når det ble tatt bilder som vi ikke klarte å lese automatisk, så var det vel basert på at du måtte ha innstallert en applikasjon på PCen din og kjøre den over en sikker linje mot en server som ligger i en annen by. Dette fungerte dårlig for enklte. Vi ligger etter og trenger flere folk, så laget en web applikasjon som gjorde akkurat det samme som de kunne kjøre hvor som helst i fra. Det var en smidig tilpasning. Vi tar den. En liten jobb for å øke funksjonaliteten. Det var et behov og så løste vi det selv om det gikk på bekostning av annen funksjonalitet. Det er klart at vi utnytter ressursene på en slik måte at det koster oss ikke så mye i prosjektet. Det var ikke på grunn av Scrum at det der skjedde (at de løste det på den måten). Det ville vi ha gjort likevel. Det var problemløsning aktivt. Minimaliser skalaen når du er fosinket. Det handlet mer om prioriteringer fra vår side, enn direkte følger av bruk av Scrum. Det var jo den smidigheten der med å kunne se det at, ja, vi kjørte Scrum med iterasjoner og slik, og vi så at det var krav i kravspesifikasjonen som aldri ville bli oppfylt, fordi kunden sa at ingen kom til å bruke det likevel. Men i og med at vi var forsinket og måtte prioritere slik, så ville det ha skjedd uansett. Vi bare skyver det fremfor oss så lenge at kunden sier det at vi trenger det ikke. I og med at det var bestkrevet fra A til Å hva det systemet skulle gjøre, så var det bare for oss å si det at vi begynner å så jobber vi utover ved å prioritere rekkefølgen. Da vil det være noe som har lav prioritet på slutten, så får vi finne ut om vi skal kutte det ut og heller ta noe annet i stedet. Det blir et kontraktuelt spørsmål. Det har heller ikke noe med Scrum kan du si.

Jeg tar bare et siste spørsmål om Scrum of Scrums møtene. Siden disse møtene ble holdt rett etter daily stand-up møtet, da betyr det vel at dere (i prosjekt-15) hadde Scrum of Scrums møte mer eller mindre hver eneste dag..?

Det var i prosjekt-15, så det skal jeg ikke uttale meg om, men jeg tror de hadde Scrum of Scrum daglig. Jeg tror det. Hvertfall ut i fra slik som jeg har hørt fra teorien er det at du har scrum møter (daily stand-up møter) og da får *scrum masteren* vite om noen har et problem.

Det er ikke sikkert at du kan løse det i ditt team, så da må du opp til neste nivå og si at vi har disse problemene. Det kan være infrastruktur. Det kan være testmiljøet som ikke fungerer og slike ting. Da tar du (som *scrum master*) det med deg opp til neste nivå og da tar du med deg en person. Det var slik at det ikke bare var *scrum mastere* (som deltok på Scrum of Scrums), men de kunne ta med seg en person med et problem for å introduseere det problemet i Scrum of Scrums møtet, har jeg forstått. Det var ikke en lukket forsamling, slik at det var kun *scrum mastere* som var der. Det var stort sett; ”Hva har teamet gjort siden i går?”, ”Hva skal teamet gjøre til i morgen?” og ”Hvilke utfordringer har vi for å nå målene våre?”. Det fungerer akkurat på samme måte, bare det at du snakker om team i stedet for person. Sånn jeg har forstått det, så var det faktisk daglig, mens meta Scrum møtet var mer ukentlig.

B.6 Interview with *scrum master* in project-14 and -15, 30.April 09

- *Hvor mange er dere i hvert team?*

Nå er det jo flere prosjekter hvor vi har kjørt Scrum, men i det prosjektet hvor vi har kjørt det lengst der var vi vel 9 stykker.

Det kan være greit dersom du ser for deg ett av de prosjektene du har vært med på slik at vi kan foholde oss kun til ett gjennom hele samtalen. Bare for å ta det først, hvor lenge varte dette prosjektet?

Det pågikk i 1.5år.

Er det lenge siden dette prosjektet ble avsluttet?

Det ble avsluttet i januar 2009.

Var dette et Scrum-prosjekt helt fra starten av?

Det var (et) Scrum (prosjekt helt) fra starten av. Det var jeg som innførte det, da jeg var med på det.

- *Hvor lenge var det vanglig at hvert team jobbet sammen? Var det mye utskiftning av team-medlemmer underveis eller var det stabile team hele tiden?*

Vi hadde vel, til å begynne med, team på 3-4 personer, så ble det utvidet og så ble det skalert ned igjen etterhvert.

Var det noen spesiell grunn til disse endringene av team størrelse?

Det var bare rett og slett at vi hadde forskjellig arbeidsmengde.

Så dere justerte team størrelsen hele tiden etter arbeidsmengden da med andre ord?

Ja. Vel ikke hele tiden. Vi hadde en oppstartsfase, der det var en del ting som måtte på plass først og da var det ikke hensiktsmessig å ha så mye folk. Men etterhvert som arbeidsmengden økte så fikk vi dratt inn flere til teamet.

- *Ser du noen fordeler eller ulemper, utover det at du får gjort mer med flere folk på hvert team? Da tenker jeg for eksempel på det å lede de ulike teamene og den typen ting.*

Dette her var jo ett team. Ulempen ved å dra inn folk etterhvert er gjerne det at gruppe-dynamikken har allerede fått satt seg litt. Jeg tror ikke at det har noe å si for om det Scrum eller ikke.

Så har man jo dette med opplæring. Dersom det kommer nye folk inn på teamet som gjerne ikke er vant til Scrum, i det hele tatt, eller hvordan akkurat dere kjører det. Hvordan tok dere for dere denne type situasjon?

Introduksjonen til Scrum gikk egentlig rimelig greit. Vi hadde veldig mange unge folk som jobbet i dette teamet, men det var ikke der utfordringen lå. Det var vel heller på domene-kunnskap at utfordringen lå. Ikke introduksjonen til Scrum som sådan.

Når vi først er inne på det; Hvordan gjorde dere selve opplæringen an team medlemmer? Sendte dere dem på kurs eller var det (eksterne) folk som kom innom og fortalte/forklarte?

Nei, jeg har *scrum master*-kurs så jeg tok den interne opplæringen med de deltakerne.

Når det gjelder scrum mastere her i bedrften deres, hvordan gjør dere vanligvis når en ny scrum master skal læres opp? Finnes det noen spesielle rutiner for det, som å sende ham på et kurs?

(Når det gjelder opplæring av) *scrum mastere*, da må vi ut eksternt hvis vi ønsker å få dem sertifisert. Det er vel ikke så veldig mange som har lov eller kan sertifisere deg som *scrum master*. Men det er eksternt. Men internt har vi noe som vi kaller for Scrum Awareness som egentlig er et kurs for de som skal være deltakere i et Scrum team.

Når det gjelder product owner, hadde dere en intern eller ekstern product owner på dette prosjektet?

Det var en intern *product owner*. Det var det.

Var det en spesiell måte den som hadde rollen som product owner ble lært opp på for å kunne tre inn i denne rollen?

Nei, han hadde ikke noen formell trening på å være *product owner*. Det gikk seg vel litt til etterhvert. (Da tenker jeg på) arbeidsformen.

- *Når det gjelder sprint-lengde, hvilken sprint-lengde benyttet dere?*

Vi hadde 4-ukers sprinter.

- *Var det noe dere beholdt gjennom hele prosjektet eller var det noen endring på sprintlengden?*

Nei, vi hadde 4-ukers sprinter. Men det var et par tilfeller i forbindelse med ferieavvikling hvor vi justerte litt på sprint-lengden. Ellers så holdt vi oss til fire uker.

- *Var det en spesiell grunn til valget av akkurat 4-ukers sprinter?*

Ikke annet enn at det har mer eller mindre blitt en de-facto standard med 4-ukers sprinter. Vi benytter oss av fire uker i utgangspunktet og det passer relativt godt inn i prosjektene rundt oss.

Var det mange team som var på dette prosjektet fra bedriften?

Nei, fra bedriften så var vi bare to stykker.

To team?

Nei, to personer.

Var det ellers mange team som var med på dette prosjektet? Jeg tenker da i forhold til koordinering for eksempel.

Det var vel tre team, men det var bare vårt team som kjørte Scrum.

Hvilken utviklingsmetodikk kjørte de andre teamene?

De kjørte etter RUP.

Var det noen spesielle utfordringer knyttet til å samkjøre de tre teamene når to av dem kjørte RUP, mens dere kjørte Scrum?

Nei, egentlig ikke. Det vil alltid være litt diskusjoner rundt når ting skal leveres og hvorvidt dette passet inn med våre sprint-lengder og sånn, men det gikk bra det.

Hvis vi ser på sprintplanleggingsmøtene dere hadde; hvordan foregikk de? Kan du fortelle litt detaljert om hvoran disse møtene foregikk?

Dette teamet var jo et test-team, som drev med test og utrulling. Vi gjorde egentlig en sprintplanlegging og en sprint review på samme dag. Første halvdel av dagen var egentlig sprint review, og andre halvdel av dagen var sprintplanlegging. Vi var avhengig av input fra utvikling, service og vedlikehold og da var det veldig greit å få satt dette til én dag. For vår del så var innholdet i en sprint å forberede tester til de løpende prosjektene som da ble levert fra utviklingsavdelingen. Det var relativt fastsatt hva som kom til å ligge i vår (product) backlog. Det var jo neste leveranse, neste prosjekt.

Dere var hele tiden avhengig av hva de andre hadde gjort på forrige sprint..?

Ja, eller det vil si de jobbet jo ikke etter sprint-metodikk de da, men vi var avhengig av å få den leveransen de kom med, den skulle jo testes etter en gitt ramme, selvfølgelig med avvik gitt at det var forskjellige kunder som skulle ha det produktet. Det var der mesteparten av arbeidet i sprintplanleggingen lå. Det å finne hvilke avvik som var nødvendige å gjøre i forhold til den standard måten å gjøre det på. Hva var det som var spesielt med denne leveransen her, og så få det inn i sprint backlogen og få det tilpasset.

Når det gjelder estimering av hvor lang tid det ville ta å gjøre det forskjellige oppgavene, når dere skulle bryte ned product backlogen. Kan du fortelle litt mer om hvordan dette ble gjort?

Det ble gjort estimering. Det vil si gruppen gjorde estimering, men vi brukte ikke planning poker eller noe slikt. Vi begynte med å gjøre tre-punkts-estimering, men etterhvert så fikk vi jo relativt gode erfaringstall, så da ble det gjerne til at man så på hva som var gjort i tidligere sprinter, og eventuelt justerte basert på kunnskap og litt synsing.

Hvilken rolle hadde du som scrum master i forhold til estimeringen? Var det helst teamet som var aktive eller ga du råd..?

Teamet var jo de som var ansvarlig for å gjøre selve estimeringen, men jeg var jo med (som *scrum master*) å kom med input og feedback i forhold til det som ble gjort.

Følte du et typisk problem kunne være at folk ikke var aktive nok eller at noen overkjørte andre mer, opplevde du noe av denne typen problematikk?

Nei, utfordingen lå vel heller i det at vi hadde et press på oss når vi fikk levert produktet fra utviklingsavdelingen, så hadde vi et veldig stort press på oss for at vi skulle være ferdig innen én eller to sprinter for da skulle det (produktet) rulles ut til en kunde. Så det var heller der utfordringen lå. Det å få redusert det til et antall timer som lot seg gjøre i praksis. Det var egentlig mer et press utenifra.

Du nevnte dette med tre-punkts-estimat. Er dette noe du kan utdype litt mer? Hva legger du i det?

En oppgave ble estimert med en min, max og most likely.

- *Nå har vi snakket litt om hvordan for eksempel sprintplanleggingen foregikk og litt i forhold til team størrelse og den typen ting. Hva ser du på som din viktigste oppgave som scrum master i forhold til måten dere kjørte Scrum i dette prosjektet? Var det noen spesielle tiltak som du følte det var viktig for deg å få på plass?*

Ja, jeg er ikke helt sikker på hva du tenker på, men en erfaring fra min side var at det aller viktigste var det å beskytte de i teamet mot alt det som kom utenifra. Flere av de som satt i teamet ble hele tiden forespurt om å gjøre andre aktiviteter i andre steder i organisasjonen.

Plutselig var det timer som ble borte. Det å prøve å gjete dette her litt, å følge opp at de (team-medlemmene) ikke ble dratt ut, og prøve å få koordinert det slik at alle slike henvendelser kom inn til meg (først). Både det jeg har fått høre fra andre og det jeg har erfart selv så vil det ofte være slik. Det er ikke ofte du klarer å få ressurser dedikert 100%. Det er alltid noen små tidstyver innimellom. Ofte så er de uoffisielle også. Det kan være noen som kommer bort for å spørre; "kan ikke du komme bort å hjelpe meg med å fikse et eller annet." Da forsvinner plutselig 1-2-3-4 timer og hvis dette gjentar seg flere gnager i uken for flere i teamet ditt, så er det plutselig en del tid som blir borte.

Hvordan kan du håndtere slike problem? For det er vel ganske vanlig..

Det er jo å prøve å dressere den organisajsonen som er rundt og hvis det er flere team og prøve å få diskusjonen på *scrum master*-nivå slik at de kan bli enige seg i mellom om at du får lov til å låne min ressurs i x antall timer. For da kan du få lagt det inn i burndown chart'et ditt og da har du hvertfall en oversikt, hvor du kan se at dette er timer som jeg ikke har tilgjengelig. Det er mye bedre å få beskjed om at denne ressursen her er det nødvendig å bruke 50% på noe annet, så får du heller legge inn i burndown chartet at han sitter bare 50%, istedet for at dette kommer som en overraskelse når man er midtveis i en sprint.

- *Når det gjelder dette med Scrum of Scrums møter, hvordan fasiliterte dere det? Hvor ofte hadde dere møtene og hvem var det som deltok?*

Som jeg sa så var det egentlig bare ett team som kjørte Scrum. Det var det test-teamet. Utviklingsavdelingen kjørte jo RUP. Men det er jo klart at vi hadde jo våre møter mellom de avdelingene, så du kan jo i prinsippet si at det ble en Scrum of Scrums. Men det ble jo et helt vanlig koordineringsmøte. I utgangspunktet hadde vi et slik koordineringsmøte på ukentlig basis mellom avdelingene, kanskje noe hyppigere i ekstreme situasjoner.

På disse koordingeringsmøtene, hvem var det som vanligvis deltok på de?

Det var jo meg fra test teamet, så var det avdelingslederen fra utvikling og så var det avdelingslederen for service og vedlikeholdsavdelingen. Internt i denne bedriften så hadde vi noe vi kalte produktansvarlig som hadde litt mere overordnet ansvar for hvilken retning produktet skulle utvikles, uten at han satt og styrte noen ressurser eller noe slikt ansvar. Men hans oppgave og hans ansvar var å definere hvilken retning det var vi skulle videreutvikle produktet. Det kunne legge føringer for flere av gruppene egentlig.

Av og til så kunne det tenkes at produktansvarlig deltok på møtene?

Ja, hvis det var behov for det, så var han også med.

Men det var altså ikke noe høyere ledelse utover det eller eventuelt utviklere som deltok på disse møtene?

Nei, ikke på de Scrum of Scrums møtene.

I forkant, da jeg satt opp disse spørsmålene så hadde jeg egentlig tenkt på et annet prosjekt (prosjekt-15), for en av de andre scrum masterne snakket litt i går om at dere hadde et spesielt opplegg på de Scrum of Scrums møtene og at dere hadde Meta Scrums møter. Jeg lurer på om vi skulle tatt inn akkurat det, for synes at det hørtes ut som interessante funn for rapporten.

Prosjekt-15:
Det var jo et veldig stort prosjekt, hvor det var 7 utviklingsscrum team og det var ett eller to bug-fiksing team i tillegg. Team-størrelsen var vel 6-7 personer og *scrum masterne* for hvert av de teamene samlet seg til en Scrum of Scrums rett etter at de var ferdige med daily scrum (daily stand-up møte). Daily scrum (daily stand-up møte) ble avholdt tidlig om morgenen. En halvtime etterpå (daily stand-up møtet) så var det Scrum of Scrums.

Hvor lenge varte et Scrum of Scrums møte?

Prosjekt-15:
Det var satt av et kvarter.

Hvis det var 7 utvikingsteam og 1-2 bug-fiksings team, var det da 9 personer som deltok på dette møte?

Prosjekt-15:
Nei. Enkelte *scrum mastere* var *scrum mastere* for flere team. Arbeidsmengden ved det å være *scrum master* var ikke større enn at det var mulig.

Hva var typiske ting som ble tatt opp på Scrum of Scrums møtene?

Prosjekt-15:
Det var alle issues som *scrum master* ikke kunne løse direkte. De ble tatt opp på disse

Scrum of Scrums møtene. Det kunne eksempelvis være dersom du hadde problemer med utviklingsmiljøet. Akkurat det var en relativt stor issue i dette prosjektet. Andre ting kunne være tilstøtende prosjekter som kanskje hadde en leveranse som man var avhengig av, men som ikke var på plass enda. Det var typisk en ting som *scrum master* ikke kan få gjort så mye med, men da kan dette tas et nivå lenger opp.

Var det utelukkende scrum mastere som var med, for eksempel product owner var ikke involvert..?

Prosjekt-15:
Nei. Det var kun *scrum masterne* som var der.

Var denne måten dere kjørte Scrum of Scrums møtene på noe dere kjørte helt fra begynnelsen av eller var det noe dere begynte med etter hvert?

Prosjekt-15:
På prosjekt-15 var jeg ikke med helt fra begynnelsen av for det prosjektet hadde pågått i 3 år eller noe slikt. Prosjektet begynte med at det ikke var så mange team og da var vel heller ikke behovet til stedet. Men jeg antar at det kom på plass etterhvert som antallet team begynte å bli ganskje stort.

Har dere gjort noen endringer på dette møtet, mens du har vært med på prosjektet? Har dere blitt kortere/lengre eller færre/flere personer med på møtet..?

Prosjekt-15:
Nei. Det ble holdt relativt stabilt. 15 minutter pluss minus noen minutter for all del, men det var fokus på at det Scrum of Scrums møte skulle ikke gjennomføres på noen annen måte enn et helt vanlig Scrum møte (daily stand-up møte).

Var det noen spesiell grunn til det? Ofte kan du høre om Scrum of Scrums møter som avholdes én gang i uken, var det en spesiell tanke bak å avholde det så ofte?

Prosjekt-15:
Tanken bak var at vi så at de hendelsene som oppsto rundt om kring i teamene var det behov for å få løst raskt. For eksempel problemer med utviklingsmiljøet eller ikke tilgjengelige tilstøtende systemer. Du kunne ikke la 2-3-4 team sitte å vente på det til neste

Scrum of Scrums møte i neste uke. Da var det bedre å ha litt hyppigere møter og heller holde dem litt kortere.

Så var det jo også noe som ble kalt Meta Scrum. Det var jeg aldri med på. Men det var prosjektlederen som hadde det med de tilstøtende andre prosjektene og kundene. Det ble avholdt omkring én gang i uken. Der ble typiske ting som ikke kunne løses innad i prosjektet tatt opp på dette møtet. Dette var et mer regulært møte. Det var hvertfall ikke hver dag, Jeg tror det var én gang i uken. Jeg vil tro at det var litt lenger varighet på de møtene.

På disse Meta Scrums møtene, var det noen spesielle representanter utover prosjektleder på dette prosjektet som deltok? Var det alle prosjektlederne for hvert prosjekt som deltok? Hvem var det ellers som deltok?

Prosjekt-15:
Jeg vet prosjektlederen på det prosjektet vi jobbet på var med. Det er mulig at hun også hadde med seg økonomiansvarlig, altså en kontroller for det prosjektet. Så tror jeg også at det ble kalt inn, ved behov, lead architect eller andre ekstra personer avhengig av tema de jobbet med.

Ville kunderepresentanter delta på dette møtet?

Prosjekt-15:
Ja, såvidt jeg vet så var kunden med på de møtene der ja.

Det virker som det var mye interessant fra dette prosjektet (prosjekt-15). Jeg ser for meg at vi kan fortsette ved å ta litt saker fra begge prosjekter. Hvis ser på dette med opplæring i forhold til prosjekt-15. Hvordan foregikk det der?

Prosjekt-15:
Dette var jo et prosjekt for bedriften vår, så der satt det stort sett bare ansatte i bedriften og veldig mange har fått dette *scrum master*-kurset eksternt. Det var faktisk noen som ble sendt på *scrum master*-kurs underveis i prosjektet. Det husker jeg. Men det var altså et eksternt kurs.

Når det gjelder product owner på dette prosjektet, hadde dere en intern eller ekstern product owner?

Prosjekt-15:
Det var en ekstern (*product owner*).

Vet du om det var noen form opplæring av denne eksterne product owneren?

Prosjekt-15:
Det vet jeg faktisk ikke.

Var det da en product owner for alle disse 9 teamene?

Prosjekt-15:
Nei, altså *product owner* her var vel egentlig mer en rolle som ble besatt av flere. Det er vel riktig å si at prosjekt-15 fulgte jo ikke Scrum 100% hele veien. De brukte elementer fra Scrum for å si det sånn.

Hvilke elementer var det som eventuelt ikke ble brukt?

Prosjekt-15:
Det som var litt modifisert, var det at du hadde en *product owner* som satt i et design team som godkjente det som var blitt gjort i løpet av en sprint. Eksempelvis fikk du en eller annen modul som du skulle utvikle, og den modulen skulle vises frem når den var avsluttet. Da var det avhengig av hvilken person i dette design-teamet som du hadde kontakt med. Da var det jo denne personen som hadde *product owner*-rollen. Andre ting som kanskje ikke var helt sånn etter boka... Estimering var faktisk gjort på forhånd. Du fikk et utkast til et estimat. I utgangspunktet så tror man at dette kommer til å ta så og så lang tid. Du skulle jo gjøre en estimering for din egen del når du startet opp en sprint, når du fikk en modul, men det var faktisk gjort estimater på forhånd.

Så på en måte kan man si at product backlogen inneholdt estimater som design-teamet hadde gjort på forhånd?

Prosjekt-15:
Det her blir litt komplekst da. Det her gjelder nok de aller fleste prosjekter. Du vil nok aldri få en kunde til å akseptere det å jobbe fullt ut etter Scrum fordi alle har kontrakter som i

utgangspunktet er basert på vannfallsmodellen egentlig, men i best case RUP. Så kontrakt og arbeidsform passer ikke helt overens. Du får aldri en kunde til å akseptere; ”Gi meg penger og så skal jeg lage det beste jeg klarer å få til.” De fleste kunder sier det at du skal ha en gitt tidsfrist og du har en gitt sum penger og du skal oppfylle et sett med krav for å akseptert denne leveransen. Sånnsett så passer ikke Scrum helt inn for å få oppfylt de kontraktstandardene som brukes. Du får et lite avvik der. Da blir det gjerne litt tilpasning av Scrum. Man sier gjerne at man bruker Scrum og man bruker hvertfall elementer fra Scrum, men 100% Scrum vil det neppe blir før man får en annen form for kontrakt. En leverandør er nødt til å gjøre et estimat på forhånd. Si at en kunde kommer med en kravspek (kravspesifikasjon); ”Dette er det jeg ønsker å få levert.” Så gir du et tilbud på x antall kroner og en eller annen dato når du kommer til å levere. Da må du ha gjort et estimat på forhånd. Estimatet kan ikke bli for grovt, som leverandør så vil du gjerne vite at du klarer å opprettholde den kontrakten for du ønsker jo ikke å tape penger heller. Dersom du gjør et for grovt estimat uten at noen har satt seg skikkelig inn i det, så risikerer du å gå på en smell. Da tilbudet ble levert inn til prosjekt-15, så satt det folk i sikkert et par måneder å jobbet med å estimere prosjektet. Det ble gjort et godt stykke arbeid der. Det var disse estimatene (som ble gjort i denne fasen) som man hadde på forhånd (å forholde seg til i planleggingsmøtene under selve prosjektet). Slik tror jeg at det vil være en stund fremover, inntil man får en kontraktsform som lar seg bedre kombinere med Scrum. Dersom du skal følge Scrum helt ut, så vil du i prinsippet gi en leverandør frie tøyler i forhold til budsjett.

- *Er det noen praksiser, spesielt for det første prosjektet, som har endret seg etterhvert som dere har brukt Scrum (nå tenker jeg ganske bredt på det). Var det noe dere sluttet å praktisere eller noen endringer dere gjorde for å tilpasse dere etter hvert?*

Nå tenker du på tilpasninger altså..

En tilpasning kunne vært at dere gikk over fra 4-ukers sprinter til 2-ukers sprinter. En annen tilpaning kan være (endring) av team størrelse eller hyppigere/sjeldnere møter eventuelt lengre/kortere møter.

Team størrelse, det ble jo justert underveis i prosjektet. Det ble det jo. Det var jo rett og slett basert på arbeidsmengde. Andre justeringer vi gjorde.. Vi hadde faktisk ikke

burndown chart til å begynne med – sånn helt i starten. Den første og kanskje den andre sprinten gikk uten at vi hadde burndown chart som ble ført.

Når vi er inne på dette med burndown chart. Hvordan gjorde dere det? Foregikk dette på en tavle eller et excel-ark eller noe annet?

Det var på et excel-ark.

Da var det hver utviklers ansvar å oppdatere dette excel-arket eller var det scrum master som gjorde dette?

La oss skille på de to tingene. Det var det prosjektet hvor vi hadde en testgruppe. Der hadde vi et excel-ark, hvor alle gikk inn å førte forbruket og gjenstående (arbeidsmengde målt i timer).

Prosjekt-15:
På prosjekt-15 så hadde man et excel-ark for hver modul som skulle utvikles, hvor man førte burndown. Så hadde man et toppnivå excel-ark som laget en slik sammenstilling for for eksempel et team, som jobbet med sine moduler.

- *Hvis vi ser på bruk av XP, for eksempel pair programming og den typen praksiser, ble noen slike teknikker brukt i noen av prosjektene?*

Prosjekt-15:
Jeg vet det ble gjort litt par programmering på prosjekt-15.

Var det til spesielle oppgaver eller i spesielle tilfeller at par programmering ble sett på som en fordel å bruke?

Prosjekt-15:
Jeg kommer på et tilfellet hvor det ble gjort og det var da det kom inn en ny person inn i et team. Han ble da satt til å gjøre par programmering med en erfaren utvikler i det teamet.

Det var for å få kunnskapsdeling i gang?

Ja.

- *Var det noen andre typiske XP-teknikker som continuous integration eller test-drevet utvikling eller andre dere benyttet?*

Prosjekt-15:
Begge de to var egentlig en del av prosjekt-15. Vi brukte jo test-driven development, hvor vi skulle skrive testene først. Vi brukte JUnit. Vi hadde en egen byggserver som sto og tuslet og gikk i et hjørne og det var blitt laget en egen versjon av den, slik at det var en stor skjerm (synlig). Dersom noen sjekket inn noe som ikke bygde der så lyste det rødt og så fikk man (opp) et bilde av han stakkaren som hadde sjekket inn denne koden. Da satt det gjerne 50 mann og ventet på at han skulle fikse den feilen. Så det var en grunn til det (dette tiltaket), for å si det sånn.

Det var jo en ganske artig og nyttig praksis for å få kvalitetskode til å bli sjekket inn, fordi ingen har vel lyst til å være den som dukker opp på skjermen og utløser det røde lyset, og får det presset på seg. Fantes det noen andre tilsvarende eller lignende praksiser?

Prosjekt-15:
Vi hadde en sjekkliste, altså en definition of done – der var det en sjekkliste på hva som skulle gjøres. Vi skulle ha en peer review. En medarbeider fra et annet team som skulle gå gjennom det du hadde gjort og se på at du hadde fulgt (alt fra) kodestandard, rammeverket..

Hvor ofte ble dette gjort?

Prosjekt-15:
For hver gang du skulle levere en modul. Du kunne ikke si at du var ferdig med å levere en modul før det (peer reviewen) var gjort. Det var en del av definition of done at den totale sjekklisten var godkjent.

Hvordan var det i forhold til kunnskapsnivået til den personen som skulle gjøre peer reviewen? Var det viktig at den personen som skulle utføre peer reviewen var på samme nivå som den han skulle sjekke koden til eller var det såpass åpenbare ting som skulle sjekkes at de ikke nødvendigvis trengte å ha samme kompetanseområde for å kunne gjøre peer review av hverandre?

Prosjekt-15:

I de teamene så var det definert et rammeverk, hvordan ting skulle gjøres. Et teknologisk rammeverk var definert på forhånd med ganske strikte retningslinjer for hvordan man skulle utføre programmering. Det eneste var vel at de som var helt ferske ikke gjorde slike code reviews, men ellers så måtte man egentlig bare avtale et tidspunkt med en fra et annet team om at du ønsket å få en slik gjennomgang. Det var, som sagt, definerte sjekklister for hva man skulle se etter. Det var egentlig bare det å få et friskt sett av øyne til å se på det som var levert. Vi hadde vel også et krav om dokumentasjon inni koden. Javadoc for eksempel. Jeg husker ikke hva det verktøyet het, men det var et eget verktøy som gikk gjennom og sjekket at du hadde dokumentert minimum så og så mange prosent av metodene dine. Så det skulle være skrevet Javadoc. Det verktøyet kunne også si fra på hvilket nivå, altså hvor nøye det skulle være. Så du kunne ikke klare å lure deg unna ved bare å skrive signaturen på Javadoc'en, du måtte ha tekst.

Par programmering eller andre teknikker ble ikke brukt i prosjekt-14, hovedsaklig fordi det var et test-team og ikke et utviklingsteam. Vi kunne ha brukt par programmering, men det gjorde vi altså ikke.

Var det noen spesiell grunn til at dere ikke benyttet par programmering i prosjekt-14?

Det var vel mer et ressurs og organisasjonsspørsmål. Prosjekt-14 var i en bedrift som var betraktelig mindre, for å si det sånn. Det var av en helt annen skala. Prosjekt-15 var såpass mye større at der hadde du både tid og ressurser til å gjøre slike ting som kanskje i utgangspunktet tar litt lengre tid, men kvaliteten blir bedre i etterkant. Men du har råd til å ta den innvesteringen up-front, mens i det første prosjektet så var ikke det mulig.

Det prosjekt-14 var altså et mindre prosjekt med større leveringspress og mindre ressurser?

Ja. Selvfølgelig du kan mase til du blir blå i ansiktet og si at "ja, men til slutt når vi gjør opp regnskapet så blir det bedre", men det er ikke alltid at vi fikk ressursene og at viljen er der til å gjøre de tiltakene (som f.eks par programmering, continuous integration) til å begynne med når man skal levere et eller annet så fort som overhodet mulig.

- *Hvis vi tar for oss det første prosjektet, det er kanskje litt vanskelig å svare på, men hvor smidige vil du si at dere var i prosjekt-14? Dersom det for eksempel skjedde en endring plutselig, var det lett å ta tak i den og tilpasse dere i forhold til det?*

Vi var vel smidige til det ekstreme, tror jeg. Det gikk vel nesten litt over stokk og stein.

Kan du utdype det litt mer?

Når du sitter i et slikt test-team som har hovedansvar for testutrulling, så har du gjerne en deadline der hvor ting skal være ferdig testet til og du skal kanskje ha akseptansetest med kunde og etter at det er ferdig så skal det rulles ut. Den ene tingen er jo organisasjonen som du er en del av som venter på å få levert, som må få pushet det ut for å kunne gå videre med andre aktiviteter. Så har du organisasjonen på kundesiden som sitter og venter på dette. Så det er nesten så ille at det nesten ikke er noe alternativ til ikke å levere på den datoen. Det må komme i mål. Da er ting som er uteglemt eller konfigurasjoner som viser seg å være feil, det måtte bare fikses for å komme i mål til de gitte datoene for det å flytte på det og fikse fikk også konsekvenser.

Hendte det for eksempel at det førte til overtid (tenker her i forhold til XP-praksisen 40hour work week) eller at dere måtte hente inn ressurser fra andre team eller lignende for å komme i mål enkelte ganger?

Jada.

- *Hvis vi ser på det samme for prosjekt-15, hvor smidige var dere her?*

Prosjekt-15:
Du hadde et høyt leveransepress der også, men det var også knyttet til kontrakt. De fleste kontrakter har som regel litt ris bak speilet a la dagmulkt og slike ting, så det skulle leveres. Men det var et såpass mye større prosjekt, at det skulle mye til at det kom inn slike ting helt fra siden. Men så klart at det til tider der også måtte legges inn litt ekstra innsats for å komme i mål, det hendte jo. Det ble såpass mange team og var en såpass stor organisasjon på alle sider med tilstøtende prosjekter og design-teamet og slike ting at du fikk ofte avdekket ting relativt tidlig. Så det gikk rett og slett mer på produksjonskapasitet.

Tror du for eksempel at Scrum of Scrums møtet dere hadde som ble avholdt såpass ofte kan ha vært med på å hindre at det kom plutselige utfordinger?

Ja, altså du fikk i hvertfall sagt i fra relativt tidlig. Ting ble varslet oppover (i organisasjonen) tidlig. Så klart, utfordringen der ble jo at, i og med at det var et såpass stort prosjekt, så blir det også veldig mye byråkrati. Det gikk for eksempel lengre perioder hvor samtlige team-deltakerne i alle team slet med utvilingsmiljøet. Men det å få gjort noe med det, i et så stort prosjekt, å skifte utviklingsmiljø eller få gjort forbedringer på det, det er liksom ikke gjort i løpet av en dag eller to, eller en uke eller to. Det krever litt mer.

I prosjekt-15, var det noen aspekter ved Scrum som dere følte at dere ble mye bedre på i løpet av den tiden dere jobbet på prosjektet? Var det noen møter som hadde en tendens til å skli ut i tid som dere klarte å korrigere etterhvert?

Prosjekt-15:
De formelle delene av Scrum som for eksempel daily scrums (daily stand-up møte) og scrum reviews ble fulgt relativt bra. Det var veldig fokus på at, spesielt det med daily scrum (daily stand-up møtet) at man skulle fysisk skulle stå oppreist og at det skulle være veldig kort. *Scrum master* (snakker om seg selv i tredje-person) mente at dette skulle gjøres på kortest mulig tid. Det er klart at *scrum master* (snakker om seg selv i tredje-person) kan ha vært påvirket av at rett etter (30 minutter etter) daily stand-up møtet så skulle han på Scrum of Scrums møtet og da kunne du ikke bli sittende å dra det ut (i daily stand-up møtet).

I prosjekt-15 brukte dere tavler i rommene? Satt dere i åpent kontorlandskap?

Prosjekt-15:
Vi satt i åpent kontorlandskap. Hvert team hadde sitt eget scrum board.

Hvilke ting var det som typisk befant seg på scrum boardet?

Prosjekt-15:
Det var påbegynte oppgaver. Vi brukte post-it lapper. Vi hadde en del av tavlen for påbegynte oppgaver for det teamet, slik at alle visste hvem som hadde hvilke oppgaver. Du kunne få flere moduler i løpet av en sprint, hvis modulene var små. Så vi hadde post-it lapper for sprint backlogen, så når en oppgave ble tatt tak i og folk begynte å jobbe med

den så ble den flyttet til et nytt felt på det scrum boardet som het ”under construction” eller lignende. Så hadde vi eget felt for issues, hvor vi brukte røde post-it lapper. Før daily scrum (daily stand-up møtet) så skulle alle ha laget slike lapper, dersom de hadde noen issues, slik at disse kunne henges opp under daily scrum møtet (daily stand-up møtet). På disse (røde) post-it lappene så ble det også påført antall timer som var gått med til en slik issue. For eksempel hvis byggmiljøet ikke fungerte og du hadde sittet og slitt med det i fire timer, så førte du det på. For da kunne du som *scrum master* hvertfall ha en liten formening om hvor mye tid som var kastet bort på ikke-produksjonsrettede ting. Så ble også burndown chartet hengt opp.

Altså en utskrift av excel-arket?

Prosjekt-15:
Ja. Vi hadde en graf som viste burndown som var hengt opp.

Fantes det noen tilsvarende praksiser med task board og opphenging av burndown chart i prosjekt-14?

Vi kjørte litt lignende praksis der ja. Vi hadde en egen vegg med pågående aktiviteter og dem som var ansvarlig (for hver aktivitet), men den var det jeg som *scrum master* som vedlikeholdt. Vi kjørte ikke post-it lapper, men vi hadde en liste med tasks som skulle gjøres hvem som jobbet med de forskjellige taskene. Så hadde vi en issue-liste som også hang der. Der hadde vi også burndown chart hengende også en overordnet prosjektplan rett og slett.

Når du sier prosjektplan, hva legger du i det?

Det var fordi test-teamet skulle ta tak i de forskjellige leveransene og så skulle de leveres til gitte datoer. De datoene som var viktige for oss for eksempel når en akseptansetest skulle gjennomføres eller når en utrulling skulle skje, det hadde vi da hengt opp på et stort brunt papir. Det var en veldig enkel plan for når de ulike leveransene og hovedaktivitene skulle foregå. Det var et forsøk fra min side på å få litt mer forståelse og helhetsbilde for de som jobbet i teamet. Det kan jo være at noen skal planlegge ferie eller av andre årsaker skal være borte og da er det veldig greit at du.. Hvis du som team-medlem bare får de små porsjonene med disse fire ukene og du ikke vet hva som skjer lenger frem, så blir det jo litt vanskelig for din egen del å vite når jeg bør si i fra om et eller annet eller når er det jeg skal

gå å prate med *scrum master*. Jo tidligere du får beskjed om sånt jo bedre er det. Det er lettere å planlegge rundt det.

- *Hvis vi prater bittelitt kort om hvordan dere gjennomførte review- og retrospectiv-møtene, så tror jeg egentlig at jeg har alt.*

Prosjekt-15:
Jeg tror vi kan ta prosjekt-15 først. Review-møtet hadde vi faktisk to forskjellige måter det ble gjort på. En metode var at vi bare satt teamet sammen og gikk gjennom rett og slett benefits and concerns; hva fungerte og hva fungerte ikke, eventuelle tiltak vi kan gjøre for å få det til å fungere. Så vet jeg at vi hvertfall ved et tilfellet hadde et samlet møte med alle teamene, hvor vi kjørte en runde med review. Det var en kjempesamling med masse mennesker.

Hvordan skilte dette seg ut fra retrospective-møtene?

Prosjekt-15:
Vi gjorde det der (review- og retrospective-møtene) i ett og samme møte. Det ble ikke skilt på de to møtene. Det var egentlig ett møte.

Når dere hadde demonstrasjon av sprint-leveransen, hvem var det som var til stedet da?

Prosjekt-15:
Det var også to-delt fordi du som utvikler hadde en demo for den som hadde rollen som *product owner* fra design-teamet. Men i slutten av hver sprint så ble det også avholdt en stor demonstrasjon hvor det gjerne kom brukere og også folk fra prosjektledelsen hos kunden og kanskje prosjektledelsen internt hos oss, hvor det ble holdt en felles demo. Det kunne være 30-40-50 stykker som kom og så på, hvor all funksjonaliteten, som kunne vises frem, som alle teamene hadde levert var samlet. Det var en form for offisiell demo.

Når dere hadde et review-møte kun for teamet, var det da teamet, scrum master og den aktuelle product owner'en fra design-teamet?

Prosjekt-15:
På review-møtet så var det kun teamet og *scrum master*. *Product owner* var aldri innvolvert i de møtene i det hele tatt.

Product owner var altså ikke involvert i verken review- eller retrospective-delen av møtet?

Prosjekt-15:
Nei, det var ikke noe skille. Det var bare ett møte som gjennomgikk..

Da demonstrerte teamet funksjonalitet de hadde laget for hverandre..?

Prosjekt-15:
Nei, jeg vet ikke helt hvordan jeg skal få til å forklare det. Vi hadde som en del av leveransemodellen, altså definition of done, så hadde du kun en én-times seanse, ikke det en gang, med den som var definert som *product owner*, hvor du bare viste frem; "Dette her er den modulen jeg har laget. Er den ok?" Da var det greit. Det var en del av det å i det hele tatt få godkjent leveransen din. Så hadde du den store demoen som ble avholdt. Det var en felles demo, når du hadde tatt alle de modulene som var blitt levert fra hver sprint ble de satt sammen for å vise dette.

B.7 Interview with developer in project-16, 6.May 09

- *Hvor lenge varte prosjektet (vi skal snakke om i dag) og hva gikk det ut på?*

Vi startet å jobbe på prosjektet i august 2007 og jeg jobbet da til nyttår. Jeg tror jeg jobbet litt i januar også. Grunnen til at jeg sluttet å jobbe på projektet var at jeg begynte å jobbe for den bedriften jeg er ansatt i nå. Dette prosjektet var for et annet konsulentselskap. Det var et av de første prosjektene med litt størrelse der jeg var med på å kjøre Scrum.

- *Hvor mange var dere i hvert team?*

Vi hadde ett team og der var vi syv utviklere og en *scrum master*, og så var det *product owneren* som var en prosjektleder som var med på morgenmøtet (daily stand-up møtet).

Så product owner var med på alle daily stand-up møtene?

Ja, mesteparten av de og den personen var også tilgjengelig når vi hadde spørsmål og sånn, og fulgte også med på arbeidet vi gjorde underveis. Så det (*product owner*) var en fagperson som vi kunne henvende oss til.

Det var med andre ord en intern product owner?

Ja. Det selskapet vi var utleid til lager programvare for shippingindustrien. Vi var innleid som konsulenter for å hjelpe dem å lage ferdig denne programvaren. De brukte mye ny teknologi og de ville innføre Scrum som prosess og litt smidige metoder generelt. Da var vi 3-4 konsulenter inne og jobbet på det prosjektet.

Siden du sier at de hadde lyst til å innføre Scrum, så var det med andre ord det første Scrum-prosjektet de kjørte?

Ja. Da nådde vi fire hele sprinter. Vi kjørte 1-måneds sprinter.

- *Var det noen spesiell grunn til valg av sprint-lengde?*

Nei. Det tror jeg ikke. Det var ganske mye som skulle gjøres generelt og det var ganske mange tusen timer det var snakk om. Det var mange som jobbet og kanskje for å klare å levere noen features etter hver sprint så var det naturlig at de (sprintene) var såpass lange.

- *Var det noen endringer av teamet underveis? Kom det flere (team medlemmer) til eller var det flere som forsvant underveis eller var teamene relativt stabile?*

Vi var fire konsulenter til å begynne med. Etterhvert så ble vi tre. Det var vel eneste endringen. Det andre var jo det at dette var et programvarehus som hadde andre produkter også, så en del av de som var inne fra kundens side og jobbet på prosjektet de måtte gjøre andre oppgaver også utenfor prosjektet. *Scrum masteren* prøvde jo å isolere det. Det var kanskje også en ulempe med å kjøre så lange iterasjoner (sprinter), det at måtte nesten hver måned bli tatt ut for å jobbe på andre ting. Andre produkter. Jobbe med feilretting og sånn som kom opp underveis. Vi prøvde å skjerme dem så godt som mulig og da var det at vi satte opp alle slike småting som akkumulerte seg og satt dem opp til én dag eller to dager. Vi fikk jo mye henvendelser, men da var det sånn at *scrum masteren* tok de henvendelsene og når det ble såpass mye at det ble nok til å gjøre alt på én dag så tok man det da. Det var ikke helt slik i begynnelsen, men vi oppdaget jo fort at disse personene (de som var inne fra kundens side) ble forstyrret på den måten. Slik at folk utenfor prosjektet, som jobbet i de andre produktene hos den programvareleverandøren, kom å spurte ressurser internt i prosjektet; ”Kan du hjelpe oss med noe?”, eller så kom det en telefon-samtale eller e-post

eller noe sånt og da fant vi fort ut at vi burde isolere dem. Da tok *scrum masteren* tak i det og fasiliterte det slik at de kunne jobbe med sine arbdeidsoppgaver.

- *Hvis vi ser litt på et typisk sprintplanleggingsmøte med estimering og den typen aktiviteter, kan du fortelle litt om hvordan dette foregikk?*

Vi brukte faktisk på det meste nesten tre dager på en sprintplanlegging. Mye av problemet var at det produktet vi skulle lage skulle være fleksibelt slik at det kunne tilpasses hver enkelt kunde og deres behov. Det var på en måte en veldressert drivkraft fra stakeholderne, fra ledelsen i bedriften. De som hadde totaleierskap i dette produktet hadde et ønske om at produktet skulle være veldig endringsdyktig vi skulle kunne tilpasse det til prosessene og datamodellene hver enkelt bedrift hadde. Det gjorde at ting ble veldig komplekst. Det var liksom ikke bare skjermbilder som skulle lages og det skulle se likedan ut. Dette er en kunde som har.. jeg tror de har 900 kunder rundt om i hele verden, i forskjellige land og kontinenter. Det la veldig føringer for det her (produktet). Det første til at når vi satt under sprintplanleggingen, så var det ofte de som hadde mest peiling på det vi skulle lage og best teknisk forståelse som klarte å være med på sprintplanleggingen. Enkelte i teamet falt rett og slett ut. De klarte ikke å se den helhetlige løsningen og det vi skulle lage, altså totalt sett. Alle problemstillingene. Det var veldig dumt. Jeg har ikke helt, i ettertid, klart å forstå hvorfor det ble akkurat slik, men jeg forsto fort at det fungerte hvertfall ikke. Teamet kunne ikke ta eierskap i arbeidsoppgavene når de ikke kunne estimere de selv. Her var det noen i teamet som satt å dekomponerte arbeidsoppgavene; ”til det her må vi ha det her” osv. for å lage en feature. Mens noen satt bare å så på dem som spørsmålstegn; ”Hva er det egentlig vi skal lage her? Hva mener dere?” Dette førte jo til at det ble mange diskusjoner rundt hvordan vi skulle løse den tekniske implementasjonen og det var også det som gjorde at vi tok så lang tid på sprint planleggingen. Jeg har prøvd å reflektere over dette prosjektet, men jeg har ikke blitt noe klokere. Det krevde veldig mye av hver enkelt i teamet og man får et klart skille på kompetansen under disse sprint-møtene (sprint-planleggingsmøtene). De som var svakest faglig og teknisk, og hadde minst forståelse for produktet som skulle utvikles, de falt helt av under sprint-møtene (sprintplanleggings-møtene). De kunne like godt har gjort noe annet.

Hvordan påvirket dette hvilke oppgaver de (som falt ut under sprintplanleggingsmøtene) fikk i etterkant av møtet og i hvilken grad de klarte å utføre dem?

Det ble slik at de som hadde mest peiling, var de som viste mest engasjement og det var de som tok de morsomste oppgavene. Da fikk gjerne de andre som hang litt etter og på en måte sto på sidelinjen automatisk de kjedeligste oppgavene eller mulighet til kun å velge mindre morsomme oppgaver. Det var veldig negativt for teamet sin del.

Var fordelingen av de som falt av og ikke falt av under møtene knyttet til dem som var med fra kundens side og dem som var konsulenter eller hadde ikke dette noe med saken å gjøre?

Det var på kundesiden de falt av. Det var en grunn til at konsulentene var leid inn. Det var for å få ekspertise inn dit. Vi så jo også veldig klart hvem det var på kundesiden som hang med og hvem det var som falt av. De var veldige sterke personligheter, de personene i teamet som hadde mest eierskap i prosjektet. Det var de som var veldig engasjert og som hadde størst forståelse. Det er kanskje naturlig, men det førte jo til at de gjorde de artige oppgavene, mens de andre som hadde falt av turte ikke å si fra. De turte ikke å stille dumme spørsmål. Det var et skille. Konsulentene var naturligvis med hele tiden. Det var jo jobben vår å forstå hva det var kunden ville ha.

De som estimerte, det vil si de som ikke falt av og altså var med på å estimere, brukte de teknikker som for eksempel planning poker?

Jeg tror vi prøvde å kjøre planning poker i starten. Ja, vi kjørte faktisk det. Det gjorde vi, men der igjen så vi med en gang forskjellen på hvem som hang med og hvem ikke hang med. De som ikke hang med kunne gjerne gi et estimat som var veldig mye lavere enn de som hang med, og et høyt estimat for ting som de som hang med så på som enkelt. Så det ble et veldig skille der.

Var det derfor dere droppet å kjøre planning poker?

Vi droppet det på de siste sprintene. Det var akkurat av den grunn at det var ikke vits å gjøre det på grunn av at hele teamet ikke hang med.

- *Hvis vi ser litt mer på generelle problemstillinger i prosjektet, på problemløsning og hvordan det ble tatt tak i. Var hele teamet med å diskuterte når et problem oppsto eller var det kanskje de som var mest kunnskapsrike eller spurte dere product owner dersom et problem dukket opp?*

Når det var et problem som måtte løses så ble det gjerne diskutert internt i teamet. Hvis det da var noen som for eksempel hadde tatt på seg oppgaven med å lage brukergrensesnitt og man oppdaget at det var et problem som måtte løses. Da var det de som hadde ansvar for oppgaven som satt seg sammen og tenkte gjennom problemstillingen og løste det sammen. Hvis de fortsatt ikke hadde klart å løse det, hvis det var et stort problem og de så at de ville kunne få input fra de andre utviklerne, så tok de det opp på morgenmøtet (daily stand-up møtet) og så satt vi opp et møte med de rette folkene. Hvis problemet ikke ble løst da, så tok man gjerne en spike etter sprinten. Vi pleide å ha noen dager mellom hver sprint til å gjøre slike ting. Når det var naturlig, så ble *product owneren* involvert. Problemstillingene i dette firmaet var ofte tekniske. Dette var fordi en av drikreftene til dette prosjektet, slik jeg sa i begynnelsen, var basert på det at systemet deres skulle være fleksibelt. Arkitekturen skulle være fleksibel. Derfor var det veldig mye tekniske detaljer som ble diskutert og lite problemstillinger som omhandlet domene. Men når det var det (problemstillinger som omhandlet domene), så var alltid *product owneren* involvert. Han satt også i samme prosjektlokalene, så det var veldig lett å gå bort å spør om hjelp. *Product owneren* var ikke hardt belastet.

Det var flere stakeholders inni bildet. Det var jo de som eide selskapet, som sitter i styret, altså ledelsen i selskapet. Vi presenterte det vi hadde laget for hver enkelt sprint og da ble de innvolvert. Da var det ikke alltid at de hadde tid og da hadde vi litt sånn luksus-problemer med å få innspill fra dem. Vi hadde en prosjektleder også som var *product owner* og da gikk det greit for han hadde allokert såpass mye tid til dette prosjektet og jobbet egentlig fulltid på det også, så vi fikk mye input fra han.

Satt dere forresten i et åpent kontorlandskap?

Det her selskapet var ikke så stort. De hadde én etasje i et bygg med åpent kontorlandskap som også var delt med de andre ansatte som jobbet på andre programvareprosjekter. Det førte til at det kunne bli litt støy. Det vi oppdaget, blant annet, var at når noen av team-medlemmene fra kundens side ble spurt fra noen på utsiden, fra andre prosjekter i samme

selskap, så kom de gjerne rett til pulten deres å spurte. Uten å tenke seg om at de ble forstyrret. Man satt gjerne og par programmerte og jobbet på noen oppgaver på prosjektet, og så blir man forstyrret av slike henvendelser. Det var én av de tingene vi tok tak i fort, ved at *scrum masteren* isolerte dem og så kom alle de henvendelsene til *scrum master* istedenfor, og at man prøvde å gjøre det (henvende seg) på e-post også. Men det var lettere sagt enn gjort, så det tok jo litt tid før de andre som ikke jobbet i prosjektet forsto at de ikke kunne henvende seg på denne måten, for sånn var det jo ”alltid blitt gjort.”

Brukte dere task board eller den typen hjelpende elementer?

Ja, vi brukte et verktøy som het Target Process. Det er et web-basert verktøy som vi innstallerte på en web-server lokalt og kjørte det. Det brukte vi under sprintplanleggingen. Det fungerte også som task board for alle disse oppgavene. *Product owneren* brukte det også for å legge inn user stories i backlogen. Det var et ok verktøy. Det som er litt morsomt er at det hadde en grid-visning som viste taskene i en sprint. Den første kolonnen, jeg husker ikke helt hva den het, men her lå alle taskene vi ikke hadde gjort noe med. Den neste kolonnnen var ”In progress” og kolonnen etter der igjen var ”Ready for test” og den etter der igjen var ”Complete”. Når vi hadde daily stand-up på morgenmøtet, så hadde vi den (grid-visningen) på prosjektoren. Når jeg da sa at; ”I dag har jeg gjort ferdig det her skjermbildet og gjort ferdig kryptering av passord i databasen”, så kunne vi sette disse taskene som ferdig eller i ”Ready for test” eller ”Done/Complete”-kolonnen. Så sa jeg at; ”I dag skal jeg begynne på komprimering av datatrafikk mellom klient og server”, da var det en task som ble satt til ”In progress” og så gjorde vi det (flyttet selve tasken). Da så alle at nå var jeg blitt ferdig med de to (taskene) og nå begynte jeg å gjøre denne her (tasken). Da ble det veldig synlig hva vi jobbet med. Så var det selvsagt å fortelle hva vi hadde problemer med, hvis vi ikke klarte å ferdigstille arbeidsoppgavene. Det var utrolig nyttig å gjøre det på denne måten. Det likte jeg virkelig godt. Det tror jeg at teamet også likte.

Var det slik at etter morgenmøtet (daily stand-up møtet) var ferdig så skrudde dere av prosjektoren og da var ikke prosjektstatusen synlig på veggen lenger?

Nei (den var ikke synlig etter møtet).

Da måtte dere gå inn i Target Process programmet for å se prosjektstatus?

Da hadde vi en nettside, som var mer eller mindre oppe hele tiden, for å gjøre re-estimering og remaining hours for hver enkelt task og sånn. Det tok jo litt tid før alle vente seg til å bruke det. Det var veldig viktig med tanke på sprint burndown at alle førte remaining hours og alle timene man jobbet. Det tok kanskje én sprint, så var det mer eller mindre innarbeidet. Det som var artig, på morgenmøtet (daily stand-up møtet), det vi sluttet det med var å vise sprint burndown. Det her er burndown grafen vår akkurat nå. Da så vi veldig fort (team) velocity'en. Det kunne vært motiverende og samtidig varsle at nå er det faktisk ganske mange timer før vi er on-track hvis vi skal klare å levere det som er forventet i sprinten. På disse morgenmøtene var jo også *product owneren* og *scrum masteren* med, og da så vi jo veldig fort om vi måtte gjøre noen tiltak.

Du nevnte at det tok ca.1 sprint før folk ble vant med å legge inn remaining hours. Et typisk problem er at istedet for å gjøre dette hver dag, så blir det kanskje bare gjort 1-2 ganger i uken etterhvert. Slik at burndown chartet ikke er fullstendig oppdatert, og da kan det være litt vanskelig å holde oversikt over hvordan man ligger an. Dette fungerte veldig bra altså?

Det fungerte bra og grunnen til det var at.. Vanligvis når man jobber som programvare-utviklingskonsulent så fører man jo timene sine et sted for at man skal få lønn og alt det der. Men det de hadde gjort, var at de hadde en database som lå i bunnen som de hadde hentet ut timene fra og eksportert dem inn i timeføringssystemet deres automatisk. Du hadde én ressurs, én plass å føre timer på. Da var det på en måte ikke denne dobbel bokføringen som man har gjort før i tiden. Det problemet var ikke til stedet lenger. Nå hadde de ett sted hvor de kunne notere de timene de hadde jobbet. Det var veldig godt dokumentert, med tanke på at det var så fin-granulert helt ned til task nivå. Det fungerte bra. Mens på det prosjektet jeg jobber på nå, så må jeg føre timer tre steder. Det er mye verre. Når det er mye å gjøre så fører jeg remaining hours kun én gang i uken. Det er ikke fordi jeg ikke har lyst til å gjøre det, men det er fordi at det er så mye å gjøre at jeg rett og slett glemmer det og fokuserer på å levere det jeg skal levere. Det tror jeg også er litt mer en show-stopper. Det at man må føre timer så mange steder. Ofte så overholder jeg ikke tidsfristene på det, fordi det er så mye arbeid. Jeg bruker kanskje en halv time til en time i uken på å føre timer, altså på å ikke løse problemer og når man allerede jobber en del overtid så blir det ganske frustrerende. Mens i dette prosjektet så var det bare ett sted å føre

timer. Men da hadde jo jeg som konsulent to steder, men jeg så jo at de andre i teamet (de som var på kundesiden) hadde det mye lettere når det gjaldt å føre timer på grunn av det. Det tror jeg er en observasjon som er verdt å ta med seg. Jeg tror også at det at vi synliggjorde burndown grafen på daily stand-up møtet var med å hjalp, for da kunne alle se at vi som team beveger oss i samme retning. Nå jobber vi oss nedover mot null-punktet på x-aksen i felleskap. Det tror jeg hjalp på.

- *Når det gjelder daily stand-up møtene, var det noen aspekter som kanskje skiller seg ut fra Scrum by-the-book? Vanligvis er det jo slik at hvert team medlem forteller hva de har gjort siden sist møtet, hva de skal gjøre til neste møtet og hvilke impediments de har. Du har også vært innom dette med burndown grafen og bruk av prosjektor hvor dere flytter over oppgaver. Er det noen andre metoder dere brukte for å synkronisere bidrag fra team medlemmene eller var det noen andre måter dere sørget for at alle hadde oversikt over hvor langt folk hadde kommet?*

Nei, det var akkurat det vi gjorde. I begynnelsen hadde vi litt trøbbel med å få det til å gå fort nok. Vi skulle jo bare bruke et kvarter, men med mer trening så gikk det bra. Det var det vi gjorde på daily stand-up og ikke noe mer. Alle var veldig åpne for å spørre hverandre hvis det var noe man lurte på. ”Hvordan fungerer egentlig det der du jobber med Gøran? Nå skal jo du komprimerer datatrafikken og sånt, kan ikke du fortelle meg litt om det for jeg sitter og gjør brukergrensesnittet.” Da satt vi i et åpent kontorlandskap, så da kom de jo bare til meg og spurte meg om det. Det var det vi gjorde på daily stand-up. Vi prøvde å gjør det veldig by-the-book i og med at det var kundens første prosjekt.

Dere var kun ett team der som kjørte Scrum, så det var ikke så veldig aktuelt med Scrum of Scrums i den sammenhengen?

Nei, fordi de andre teamene holdt på med andre programvareprodukter, så det var ikke aktuelt der og da. Det var også slik at de skulle få innarbeide Scrum som prosess. I 2007 så fikk Scrum veldig mye medieomtale og det med smidig var da og er fortsatt veldig i vinden. Dette var det første prosjektet hvor de skulle prøve å kjøre det på. Jeg ser ikke bort i fra at de nå kjører Scrum på andre prosjekter uten at jeg vet det for sikkert.

- *Når det gjelder bruk av XP-teknikker, du nevnte par-programmering såvidt tidligere, var det i helt spesielle sammenhenger at dere brukte par-programmering*

eller var det kun i enkelte tilfeller dere følte at det ikke passet eller var det lagt opp til at dere skulle bruke denne teknikken? Hvordan fungerte dette?

Nei, altså det var ikke slik at vi måtte par-programmere, men noen gjorde det når det var vanskelige implementasjoner som skulle gjøres. Det er jo litt hyggelig å jobbe sammen på den måten også. Det er fin teknikk for kompetanseoverføring og det er en fin måte å holde oss veldig fokusert på. Det var en person i det teamet som jobbet for kunden, en mann i 40-årsalderen, som vi hadde veldig problemer med. Jeg tror at faglig så ville han veldig mye, men han slet med å gjøre arbeidsoppgavene sine. Jeg vet ikke helt hvorfor han gjorde det. Denne her personen var det veldig vanskelig å par-programmere med. Det jeg prøvde å gjøre når jeg så at han ikke klarte å gjøre oppgavene sine, det var å spørre ham om han trengte noe hjelp eller om han ønsket å par-programmere. Da ville han på en måte aldri gjøre det, og jeg kunne heller ikke tvinge meg på. Jeg tror kanskje at litt av hans problem var at vi kom som outsidere og han hadde kanskje litt frykt for å ikke prestere og det ville vises veldig fort hvis han ikke klarte å levere – at han ikke var god nok. Så han var det problematisk å par-programmere med. Det var ikke slik at det var gitt retningslinjer for å par-programmere, men samtidig var det ikke slik at ledelsen kunne blande seg inn og si; ”Nå ser jeg at dere sitter to stykker foran en datamaskin, er det særlig effektivt?” Så det var ingen som stilte spørsmålstegn ved det. Hvis vi var to stykker som par-programmerte i 2-3 dager eller én uke, så var det ingen som kom og sa; ”Nå må dere sette dere på hver deres datamaskin.” Det har jeg opplevd - at det kan skje. Jeg har fått beskjed om at det ikke er riktig bruk av ressurser. Men på dette prosjektet så var det helt åpent. Det var mer slik at vi gjorde det (par-programmerte) om vi ville. Men som sagt jeg prøvde nok kanskje mest å få dette til med den kanskje svakeste personen i prosjektet, for jeg ville ha opp den totale kompetansen i teamet. Jeg prøvde å par-programmere med alle for å lære det ut for alle de som var ansatt i bedriften var nye for test-dreven utvikling. Da kjørte vi litt sånn kursing og kanskje en workshop midt inne i en sprint, der vi snakket om test-dreven utvikling og prøvde oss litt frem på det. Jeg prøvde ofte å par-programmere med de i teamet som var nye på test-dreven utvikling, sitte ved siden av dem og la dem skrive, og prøve å observere hvordan de jobbet og håndtere det. For vi skulle gjøre test-dreven utvikling. Det var veldig vanskelig å omstille seg. Det tok noen måneder før de (som var ansatt i bedriften) syntes det var ok å skrive tester. I begynnelsen tok det mye mye lengre tid, syntes de. Da tror jeg at det var greit å par-programmere. Da hadde de noen som satt å så dem over skulderen og det ble ikke noen digresjoner eller prokrastinering.

- *Continuous integration, ble den typen praksis brukt i prosjektet?*

Ja, vi kjørte continuous integration, men det var med litt blandede følelser. Jeg har alltid hatt klokkeklar tro på det og jeg vet at det alltid fungerer, men det var en del personer, fra kundens side, i prosjektet som var skeptiske til det. "Er dette egentlig vits? Hva får vi ut av det?" Det er naturlig å være skeptisk når du aldri har prøvd det før. Det tok jo litt tid å få det til å kjøre. De startet noen uker før meg siden jeg var på sommerferie. De hadde satt opp continuous integration slik at denne prosessen hentet den siste kildekoden, kompilerte og kjørte enhetstestene. Bare dette er jo veldig verdifullt. Men det jeg har oppdaget når vi kjører contiunous integration er at det er veldig kult hvis man etter at man har kompilert koden og kjørt enhetstestene, kan deploye dette her til et testmiljø som alle i prosjektet og kunden har tilgang til slik at man hver dag eller hver time om man vil kan gå inn å se hvordan produktet ser ut nå. Da får *product owneren* hele tiden et bilde på tilstanden til den programvaren vi lager. Når jeg hadde gjort det, fikk jeg inntrykk av at alle i teamet virkelig så verdien i å ha continuous integration fordi nå trengte de ikke å deploye til testmiljøet en gang. *Product owneren* trykket bare på et ikon på skrivebordet sitt og så innstallerte siste versjon seg og så kjørte han siste versjon av programmet. Vi brukte click-once deployment. Vi laget en Windows Presentation Foundation (WPF) applikasjon, en Windows app. *Product owner* så en enorm verdi av det og da først, kanskje én måned siden vi hadde startet, så forsto han hvorfor vi skulle benytte continuous integration. I utgangspunktet forsto han (*product owner*) ikke hva enhetstesting var en gang, han visste bare at det gjør koden litt bedre. Men når han også kunne få return on investment (ROI) på continuous integration, så syntes han at det var lurt å putte noen timer i hver sprint til å vedlikeholde denne prosessen. Det er slik med de vektøyene som finnes på markedet i dag, at det tar litt tid å vedlikholde continuous integration prosessen. Jeg har samtidig hatt prosjekter ute hos kunder der vi har kjørt helt vanvittige continuous integration prosjekter. Hvis du vil høre mer, bare som en digresjon, så kan jeg fortelle om det eller så kan vi gjøre oss ferdig.

- *Vi kan gjerne ta det etterpå, men det hadde vært spennende å få høre mer om det. Vi kan ta litt kort om review- og retrospektivmøter. Hvilke praksiser hadde dere der, hvem deltok og hva var typisk lengde på møtene?*

På retrospektivene (retrospektivmøtene) brukte vi 2-3 timer. De som deltok var teamet, ikke *product owneren*, men teamet og *scrum masteren*. Det vi gjorde var at alle fikk post-it lapper og da skrev man på dem hva som ikke fungerte. Så hadde vi brun-papir på veggen,

så klistret vi opp hver sin lapp og så fortalte vi at det her fungerte ikke og så fortalte vi hvorfor det ikke fungerte og så diskuterte vi det litt i teamet. Så var det neste lapp. Så sto jeg og klistret opp lapper til jeg var ferdig, og så kom neste mann. Ofte så var det samme problemer som var skrevet ned på post-it lappene og da var det jo ikke vits i å diskutere noe som allerede var diskutert. Samme prosess var det på det som fungerte. Så tok vi en oppsummering hvor vi trakk ut essensen av hva som fungerte og hva som ikke fungerte. Så føler jeg litt at vi ikke gjorde noe mer med det. Vi snakket om det og vi ble bevisst på det og vi gjorde kanskje noen tiltak i neste sprint med at, la oss si at vi skulle bruke enda mindre tid på sprint planning. Da gjorde vi det, men vi dokumenterte ingen plass det vi hadde skrevet på post-it lappene, og hengt på brun-papiret, som en huskeliste til oss selv. Det ble helst til at vi gjorde noe med det vi husket i neste sprint som for eksempel å kutte ned sprint planning og mer par-programmering, test-driven development og sånn. Det brukte vi 2-3 timer på og det synes jeg fungerte bra, men det som vi feilet litt på, det var at vi en eller annen plass skulle fått en huskeliste – dette her fungerte og dette her fungerte ikke. Kanskje vi bare skulle latt brun-papiret og post-it lappene henge igjen på veggen, slik at vi så det..? For vi glemte jo etter en uke hva det var som fungerte og ikke fungerte. Det er liksom det aller aller viktigste (man husker).

- *Når det gjelder review eller demonstrasjon for kunden, hadde dere noen spesielle måter dere gjorde det på?*

Det som var spesielt med dette prosjektet var at det var kunden som hadde et krav om dette skulle være en fleksibel arkitektur. Dette her produktet skulle skreddersys og tilpasses hver enkelt kunde. Det var liksom ikke bare å lage noen skjermbilder og si at: ”Vi har implementert disse featurene.” Av og til så måtte de også se på koden vår. Etter hver sprint, etter at vi hadde kjørt retrospektivmøtet, så viste vi det vi hadde gjort for kunden. Da var det bare å trykke på dette ikonet på skrivebordet til *product owneren* og kjøre denne applikasjonen. Dette gjorde vi gjerne på en prosjektor og så viste vi funksjonene vi hadde implementert. Deretter snakket vi om hvordan vi lå an i løypen med tanke på å implementere den fleksibiliteten de (kunden) ønsket med å endre prosesser og endre data strukturer og modeller i datasystemet. Da la vi frem det vi hadde gjort og så ga de (kunden) tilbakemelding på det og så stakk vi fingen i lufta og kjente om vi var på riktig vei eller ikke. Så gjorde vi de justeringene vi trengte etter det. Det fungerte bra. Det som var litt spesielt med selskapet var at det hadde fire eiere og alle disse satt i ledelsen, men det var

ikke alltid at alle disse var tilstedet på det møtet. Det betydde jo at at vi ikke alltid fikk all den inputen vi trengte.

Når dere hadde disse demonstrasjonene, så var det teamet, scrum master, product owner og forhåpentligvis disse fire personene fra ledelsen som deltok?

Ja, det stemmer. Det var en fordel og ha teamet der for det ble en del diskusjon og da fikk jo alle være med. Men da var det igjen det problemet med at det var kun de som var mest aktive i teamet som diskuterte med ledelsen, mens de som vanligvis satt med hendene i lommen langt nede i stolen, satt slik på disse møtene også.

Hadde dere noen akseptansekriterier som var definert på forhånd som dere jobbet mot, slik at dere kunne se når dere var ferdige og når demonstrasjonen kom så burde de dermed være fornøyde med resultatet?

Ja, altså vi hadde jo (product) backlog'en og i den så hadde vi et sett med user stories. Når vi viste dette til ledelsen så kjørte vi ikke gjennom noen akseptansetester og vi hadde ikke noen formell prosess på det. Vi sa bare: "Dette har vi gjort ferdig, dette har vi ikke nådd å gjøre ferdig". Vi jobbet jo hele tiden med å få dette systemet fleksibelt nok. Men det syns jeg manglet. Vi hadde ikke en sign-off, der vi sa: "Ok, nå er vi ferdig. Nå har vi levert det vi skal levere. Nå har vi implementert denne her user storien." Det hadde vi ikke.

Jeg har hørt om team som har hatt f.eks interne wiki-sider og den typen hjelpemidler vedrørende ting som de har funnet som kanskje kan være smarte løsninger på problemer slik at andre som opplever de samme problemene bare kan gå inn å lese der. Hadde dere noen tiltak som kan minne om dette?

Ja, vi hadde et diskusjonsforum der vi tok opp slike ting. Jeg skulle ønske det var en wiki og jeg har gjort prosjekter i ettertid der vi har brukt wiki og det fungerer veldig bra. Det vi kaller det nå er en utviklerhåndbok. Der vi skriver hvilken code-style vi forventer, hvordan man henter ting ut fra kildekodekontrolleren, hvordan man commit'er til den, litt om test-dreven utvikling, lenker til artikler om test-dreven utvikling, hvilke verktøy vi bruker, linker til hvordan man bruker verktøyene osv. Det kaller jeg for en utvikler-håndbok. Den er i form av en wiki. Det er noe jeg har tatt med meg som jeg kjører i alle prosjekter. Det er å lage denne håndboken.

Det er lurt å akkumulere opp slike fornuftige praksiser.

Ja, det jeg ønsker neste gang jeg kommer ut i et prosjekt, er at jeg lager en håndbok jeg på en måte kan ta med meg videre til et nytt prosjekt igjen, der ting er nesten helt likedan. På et prosjekt jeg sitter på nå, så har de en svær slik håndbok. Jeg var med på å lage den da de startet opp prosjektet og ga innspill på den. Det (en slik utviklerhåndbok) er veldig vanlig.

- *Kommer du på noen andre informasjonskilder som var viktige for teamet, utenom product backlog, sprint backlog, user stories og dette diskusjonsforumet dere hadde? Var det noen andre steder dere fikk informasjon fra som påvirket hvordan teamet jobbet?*

Ja, vi hadde en tavle på veggen. Et whiteboard. På det whiteboardet skrev vi remaining working days left av sprinten. Det oppdaterte *product owneren* hver dag. Så hadde hadde vi et nummer til på den tavlen og det tror jeg var antall dager igjen til første release. Disse tallene ble oppdatert hver dag og det hadde en veldig positiv effekt, hvertfall dette med antall dager igjen av sprinten. Da ble man litt ekstra motivert når dette tallet begynte å minke. Det jeg savnet var en sprint burndown som ble oppdatert kontinuerlig og plassert i kontorlandskapet.

En annen ting som var en viktig informasjonskilde var de bøkene utviklerne i teamet leste.

Var dette (bøkene) noe som lå rundt i lokalet eller noe som utviklerne tok med seg selv?

Nei, det var litt sånn at vi som kom inn som konsulenter anbefalte noen bøker til dem (de andre utviklerne på kunde-siden) og så leste de dem, og så ble det litt snakk og sånt om det. Det var veldig nyttig sånn, men det var mer teknisk.

Når det gjelder trening og opplæring av de forskjellige team medlemmene, og nå kom jo dere inn som konsulenter og kunne Scrum ganske bra, mens kunden gjerne ikke kunne dette så veldig bra. Ja, bare for å ta det; var scrum master en person som stilte fra deres side?

Nei, det var en person fra kundens side.

- *Hvordan foregikk opplæringen av den personen som skulle være scrum master?*

Han (scrum master) ble opplært før jeg kom inn i prosjektet. Prosjektet startet noen uker før jeg kom inn og jeg tror det var en som var Scrum-sertifisert fra oss (fra konsulent-siden) som tok opplæringen med ham. Han jobbet også i teamet, så det var en kontinuerlig opplæring av denne *scrum masteren*. Ideelt sett så skulle han ha vært på et Scrum-kurs, men det tror jeg ikke at det ble tid til. Så spurte han (som hadde rollen som *scrum master*), den personen som var Scrum-sertifisert når han lurte på noe.

- *De andre team medlemmene, de som var på kundens side, ga dere de et slags introduksjonskurs eller fortalte dere dem hvordan Scrum skulle foregå på en annen måte?*

Jada, det var et introduksjonskurs i starten (av prosjektet) og jeg tror det ble brukt noen dager på det. Etterhvert, da vi hadde kjørt to sprinter og enkelte ting fungerte og enkelte ting fungerte ikke, så fikk vi inn en prosjektleder som jobbet for det konsulentselskapet jeg jobbet i, som jobbet ute hos en annen kunde der de hadde kjørt Scrum med stor suksess. Han kom å fortalte om hva som fungerte hos dem. Det var en veldig bra måte å få i gang kompetanseoverføring. Jeg tror faktisk det var to stykker som var der å pratet om hva som fungerte og hva som ikke fungerte. Det er hvertfall det jeg husker som mest lærerikt.

Det var en stund der ting gikk litt trått og noen mistet litt troen på Scrum. Da var det bra at de kom å motiverte.

Er det noen utfordringer du ser, kanskje spesielt i dette prosjektet, som vi ikke har vært innom som går på prosjektledelse? Noen ting som var vanskelig når det gjaldt å styre dette prosjektet?

Jada, vi var jo innom det med at de som jobbet på prosjektet fra kundens side ble forstyrret. Det snakket vi om. Jeg nevnte han 40-åringen som vi hadde litt problemer å jobbe sammen med. Vi klarte ikke å par-programmere med ham og sånn, men vi hadde også problemer med ta han ikke kom i tide til daily stand-up, selv om vi kjørte dette rundt kvart over ni, halv ti. Altså ikke så veldig tidlig på morgenen. Han hadde jo flere unger og det kunne jo av og til forsinke ham. Vi prøvde jo mange ganger underveis å ta en prat med ham, altså vi fra utsiden (konsulentene) prøvde å sette oss ned å prate med ham. Det reagerte han veldig dårlig på. Da hadde han blitt utrolig sint og sagt; ”Hva er det dere tror dere kommer for?” Den klassiske (responsen) og begynt å forsvare seg selv og sånn. Da måtte vi rett og slett bare backe ut og ikke gjøre noe mer med det, men fortalte jo om ham til *product owneren,*

han som var prosjektleder om disse problemene og at de måtte gjøre noe med det internt. Men det ble ikke gjort så mye med det. Det var et problem vi hadde i alle sprintene han var der. Etter mitt øyemål så mener jeg at han ikke var egnet til å jobbe i prosjekt og kanskje ikke egnet som en programvareutvikler. Han kunne sikkert gjort andre oppgaver veldig bra, men å jobbe i det prosjektet vi jobbet på skulle han ikke ha gjort. Vi hadde et samarbeidsproblem med ham i fire sprinter. Der det var støy rundt ham hele tiden. Det sa vi ifra om til kunden, men i og med at vi var leid og sånn og vi sa ifra det vi kunne og de gjorde ikke noe særlig med det, så var det vanskelig for oss å gjøre noe særlig mer. Vi var jo der for å hjelpe dem og egentlig ikke for å skape noe splid og uro internt.

- *Nå har vi egentlig vært innom ganske mye, men hvis vi bare tar noe sånn helt generelt. Er det noen ting vi ikke har vært innom når det gjleder endringer av måten dere kjørte Scrum på i løpet av de fire sprintene prosjektet pågikk? Var det noen spesielle aspekter ved Scrum som ble modifisert for å passe til bedriften, som vi ikke har nevnt? Det trenger ikke å være det..*

Dette var et prosjekt hvor Scrum var nytt for kunden. Vi prøvde å gjøre det etter boken, men samtidig så prøvde vi å være pragmatiske – vi gjorde det som fungerte. Vi gjorde ingen justeringer. Jeg føler at prosessen fungerte tålig bra. Der var det ikke så mange problemer. Det var mer problemer når det gjaldt forhold til personer i prosjektet.

To veldig korte spørsmål så er vi ferdige. Det ene går på hvor smidige du hadde inntrykk av at dere var i dette prosjektet. Dersom det kom for eksempel en veldig uventet endring, i hvor stor grad følte du at dere var i stand til å tilpasse dere det?

Jeg følte at vi ikke var smidige nok. Én av grunnene til det var at ledelsen hadde veldig klare linjer på hvordan vi skulle implementere løsninger for at de skulle være fleksible nok. Jeg var én av dem som prøvde å fortelle dem at den løsningen de hadde skissert kanskje ikke var den beste, men de var veldig bastante på at det skulle gjøres slik, og da fikk ikke vi løst dette som et team. I og med at løsningen ikke var god nok eller kanskje at vi ikke forsto den slik de egentlig ville ha det, så klarte ikke vi å være smidige nok. Vi var veldig utsatte for endringer. Det tror jeg hadde litt å gjøre med teamet også. Vi måtte nok ha kjørt en del sprinter til før alle (i teamet) ble gode nok på test-driven development. For skal du være smidig, skal programvaren din være smidig og skal du være smidig for endring så må du gjøre test-driven development. Da kan du virkelig ha kode som kan endre seg uten at

det byr på noen særlig problemer. Der var vi ikke, men jeg så at det gikk bedre for hver sprint som gikk så klarte vi å bli bedre som team på test-dreven utvikling og det ga oss litt mer selvtillit. Det å kunne gjøre endringer når de kom underveis. Men når det er sagt, så opplever jeg svært svært sjelden at man er smidig nok i et prosjekt. Det skal så mye til og da må man kjøre veldig mange teknikker fra ekstrem programmering. Du finner veldig få team i Norge i dag som behersker det.

Det å være smidig er vel gjerne et mål som man sikter mot, men som det gjerne aldri vil være mulig å nå fullt ut.

Ja, jeg føler at det er litt sånn. Problemet i dag når vi lager software er at som oftest er vi presset av tid ved at vi skal levere veldig mye på kort tid. Når vi har levert så bruker vi tid på å få det til å fungere. Da har vi på en måte sydd sammen et lappeteppe, et korthus. Da tør vi ofte ikke å begynne og rikke på disse kortene fordi vi er redde for at alt skal falle sammen og enda mer skal slutte å fungere. Når det er sånn så klarer vi ikke å være smidige, og sånn er det i veldig mange prosjekter. Det er få prosjekter jeg har jobbet på der det ikke er sånn. Det er svært vanskelig.

Det blir gjerne naturlig slik i og med at programvaren blir såpass kompleks ettersom kodebasen øker i omfang og da sliter du gjerne med å få oversikten i og med at det er såpass komplekst noe som gjør deg redd for at dersom du gjør en endring så vil det påvirke for mye som du ikke har styring over. Det er da det gjerne er viktig å ha en veldig god praksis på test-dreven utvikling slik at du får se hvordan disse endringene propagerer gjennom resten av systemet.

Ikke sant. Exactly. Så hvis jeg gjør en endring i en bitteliten del av systemet og er litt usikker på hvordan det innvirker på hvordan klienten kommuniserer med serveren, hvis jeg da ikke kan kjøre en regresjonstest eller test-suite og verifisere at det jeg har gjort har ihvertfall ikke ødelagt gammel funksjonalitet, så vil jeg mest sannsynlig la være å gjøre den endringen. For da vil jeg føre en argumentasjon med meg selv der jeg vil si at det vil kanskje være mer kostbart å gjøre den endringen fordi jeg må fikse feil jeg introduserer og jeg vil bruke mye mer tid på det, for jeg vet ikke om jeg lager noen feil. Det er klassisk. Jeg tror ikke jeg har pratet med noen der det ikke har vært et problem i et prosjekt i dag.

- *Jeg bare følger opp den aller siste biten. Du nevnte at dere i løpet av prosjektet ble mye bedre på test-dreven utvikling etterhvert som tiden gikk. Var det noen andre deler/aspekter du så at dere hadde en stor fremgang på i løpet av prosjektet?*

Vi ble jo mer sammensveiset som et team. I starten var det litt sånn at alle jobbet i hver sin retning og trakk tauet i hver sin retning og vi gikk ikke fremover, men etterhvert så ble vi mer sammensveiset. Vi jobbet samlet og trakk tauet i samme retning. Vi klarte å levere mer. Det var positivt og jeg føler at alle fikk øynene opp for continuous integration og så at det var viktig, Vi ble bedre på å føre timer. Vi ble bedre på å par-programmere ettersom vi ble bedre kjent. Jeg tror også at folk ble teknisk bedre fordi et av målene var kompetanseheving da vi ble leid inn, og dette var noe vi kjørte veldig hardt på. Jeg tror at alle lærte mer og ble flinkere teknologer og flinkere programmerere.