NTNU
Norwegian University of
Science and Technology

In The Scrum

An Ethnographic Study Of Implementation and Teamwork

Øyvind Kvangardsnes

Master of Science in Informatics
Submission date: February 2008
Supervisor: Torgeir Dingsøyr, IDI
Co-supervisor: Nils Brede Moe, SINTEF

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

Abstract

Agile software development have in recent years been widely accepted in industry, as well as being the target of much research. XP has been the main focus, while there exist relatively few studies of other Agile methods such as Scrum.

This thesis describe an ethnographic study of a Scrum team in a project. The goal is to give a rich description of the use and application of Scrum. Special attention is given to the implications of differences in implementation from theory. Another focus is to reveal the dynamics of teamwork within the project.

The main findings are that Scrum was easy to implement, and worked well, but is challenging when used to increase predictability. With regard to teamwork, Scrum supported a shared mental, communication and adaptability. The leadership function is however complex, and requires good interpersonal skills.

Keywords: Agile, Scrum, Teamwork, Implementation, Software development, Software engineering

Preface

As the final step in a 2 year Masters program in Informatics, this thesis represents a full year of work.

For my part the process started with sending an innocent e-mail 30th of October, 2006 to my supervisor, asking for a short meeting to talk about a possible assignment on Agile development methods.

My initial knowledge on the subject was very basic. None of the units previously studied had anything in the curriculum on Agile methods, focusing more on the traditional side of software engineering. I was however interested from the start, because it appeared to be an interesting and evolving part of the field.

After choosing to accept this assignment, the following months were used to read articles and theory, meet with my, attend a course on Scrum in Oslo, and finally, in March starting the study. The study was finished in the end of 2007, and the thesis delivered February 6th, 2008.

In the study, my co-researcher was already involved on site using ethnographic methods. In order to better cover the project, we decided to share research material and cooperate as much as possible. This worked, for my part, extremely well as I got someone with more experience to learn from and discuss with.

Through the process, there were a number of people that contributed to the finished product you are now reading.

First and foremost the developers and people the study site deserves a honorable mention. This thesis would never had possible without your active support. Thank you for always making me feel welcome and being forthcoming at every opportunity. I'm glad you finally got a better coffee-machine, you deserve it!

Thanks to DNV Software for letting me attend their Scrum-course. This was a valuable experience.

Researchers at SINTEF ICT have all been friendly and interested in my

work, but more significantly has allowed me to play in their football team. The Monday sessions, the matches, and especially the exceptional last match of the season with me as goalkeeper will be remembered.

My class mates deserves a mention for putting up with my endless chatter on Agile methods.

My beautiful girlfriend deserves special gratitude. Supporting me when not working, and making me relax after several exhausting days of writing made my life better. Thank you.

Lastly, I would like to thank my supervisor Torgeir Dingsøyr, NTNU/SINTEF and co-researcher Nils Brede Moe, SINTEF who have both been invaluable, giving constant coaching and assistance far exceeding my initial expectations on the level of supervision.

These two have never treated me like a clueless student and made me feel like part of a team. Small things such as inviting me to lunch or coofee-breaks were greatly appreciated. Perhaps slightly more important for the study was that they gave me the opportunity to participate in the conference, CAiSE 07, where one area of focus was Agile methods. Discussing with experienced researchers on the subject was interesting. I'm

NTNU
Trondheim, February 6th, 2008

_____

Øyvind Kvangardsnes

Contents

1 Introduction **1**
- 1.1 Research Questions And Focus . . . 1
- 1.2 The Ideal Spectator . . . 2
- 1.3 Structure And Naming . . . 3

2 The Game: Rules And Tactics **4**
- 2.1 Software Engineering . . . 4
  - 2.1.1 A Brief History Of Methods . . . 6
- 2.2 Traditional Methodologies . . . 8
  - 2.2.1 Traditional Methods . . . 9
  - 2.2.2 Issues . . . 13
- 2.3 Agile Software Development . . . 13
  - 2.3.1 Motivation And Background . . . 15
  - 2.3.2 Agile Methods . . . 15
- 2.4 Comparison: Agile VS. Traditional . . . 20
- 2.5 Scrum . . . 22
  - 2.5.1 The Fundamentals . . . 23
  - 2.5.2 Studies . . . 28
  - 2.5.3 Scrum And The Big Five . . . 29

3 The Seating Arrangements **34**
- 3.1 Methodology . . . 34
  - 3.1.1 Interpretative Research . . . 35
- 3.2 The Study . . . 36
  - 3.2.1 A Choice: Research Approach . . . 36
  - 3.2.2 Fieldwork: Style Of Involvement . . . 37
  - 3.2.3 Data. Lots Of Data . . . 38
  - 3.2.4 Use Of Theory . . . 40
  - 3.2.5 Analysis . . . 41

4 Match Day: Live Scores **45**
- 4.1 Background . . . 45
  - 4.1.1 The Development Department . . . 45

4.1.2 The Project 49
4.2 Project Overview 54
4.2.1 Pre-Game: February To April 57
4.2.2 Everyday Work: April To October 59
4.2.3 Emergency Scrum: October To Christmas 61
4.2.4 Post-Game: January And Beyond 64
4.3 Applying Scrum 64
4.3.1 Estimating 64
4.3.2 Planning 65
4.3.3 Sprinting 69
4.3.4 Adapting 83

5 Results **85**
5.1 Scrum'ed 85
5.1.1 Compliant 86
5.1.2 Deviations And Adaptations 88
5.1.3 The Challenges 89
5.2 Teamwork 91
5.2.1 Coordination Mechanism 93
5.2.2 Team Leadership 94
5.2.3 Mutual Performance Monitoring 95
5.2.4 Backup Behavior 95
5.2.5 Adaptability 97
5.2.6 Team Orientation 98
5.3 Validity, Evaluation and Justification 98
5.3.1 Seven Principles 98

6 The Final Table **101**
6.1 Impact of Scrum in the project 101
6.1.1 Implementation 102
6.1.2 Teamwork 103
6.2 Implications For Research And Practice 104
6.3 The Next Matches 105

A Quantitative Data **110**
A.1 Burndown Charts 110
A.2 The project in numbers 115

B Research templates **118**

List of Figures

2.1 The iron triangle 5
2.2 Evolution of Software process models 6
2.3 The main steps of traditional software engineering 8
2.4 Royce's initial lifecycle model 10
2.5 The spiral model 11
2.6 Rational Unified Process 12
2.7 The XP Life cycle 16
2.8 Crystal methodologies 18
2.9 The planning spectrum 20
2.10 The Scrum Process 22
2.11 Life-cycle support of various Agile methods 23
2.12 Sample burndown chart 25
2.13 The "Big Five" of teamwork. 29

3.1 Distribution of contacts per week 39
3.2 Analyzing with post-its and whiteboard. 43

4.1 Organizational chart 46
4.2 "Emergency center" 47
4.3 Different projects in the company 50
4.4 Experiences from a retrospective 51
4.5 The teams "island" in the development area 54
4.6 Office floor plan 55
4.7 Timeline of the project 55
4.8 Interface design using whiteboards 58
4.9 Sprint 1 Burndown Chart 60
4.10 Sprint 4 Burndown Chart 60
4.11 Sprint 6 Burndown Chart 62
4.12 Estimation meeting prior to sprint 3 65
4.13 The product backlog 66
4.14 Cartoons next to the index card wall 72
4.15 Daily Scrum meeting in the emergency phase 74
4.16 Retrospective meeting held after sprint 3. 76

4.17 Finished-section of the index card wall, week 45. 77
4.18 Screenshot from the product backlog in Visual Studio 79
4.19 The index card wall in week 35. 80
4.20 Index cards 81
4.21 Manual burndown chart 82

5.1 The "Big Five" in the project 92

List of Tables

2.1 Comparison between Traditional and Agile methods. 21

3.1 Distinct contacts through the study. 40

4.1 Key actors in the project 52
4.2 Context factors in the study 56

Chapter 1

Introduction

Computers, and computer software has in the last decades become an indispensable part of modern life. Basically every high-tech gadget from your mobile phone through everything on the internet to the new Airbus A380 is completely dependent on quality software.

Despite this, most people has probably experienced faulty or difficult to use software. It is not uncommon that large software projects fail, with subsequent outcries in press. This is by some attributed to a lack of maturity in the software industry.

Agile methods are a "new" approach to creating software, which has been widely adopted in industry. These claim to improve on the traditional approaches to software development process, improving adaptability, predictability and effectiveness.

1.1 Research Questions And Focus

The initial research question for this study is the following:

> RQ: Present an in depth, rich description of a Scrum project.

Agile methods are still relatively young, and there exists relatively few scientific studies on the use of Scrum Dybå and Dingsøyr [2008]. This research questions aims to add to existing research on Scrum, and give insight into problems and opportunities in a project using this method. How the different practices

In addition to this rather wide research question, two additional and more specific ones:

RQ 1: How is Scrum implemented in a project compared to the book? What are the benefits and disadvantages?

Traditional methodologies describe large numbers of complex practices and techniques, which are seldom implemented as advertised [Avison and Fitzgerald, 2003]. Scrum literature also describe a number practices and techniques, but compared to traditional methodologies, the number of practices are relatively low. RQ1 aims to illustrate how Scrum is implemented in an actual project, and highlight whether or not the practices are used in a fashion similar to theory.

RQ 2: How does Scrum support teamwork?

The self organizing team involved in the entire development process is an important philosophy in Scrum [Schwaber and Beedle, 2001]. This is different from traditional methods where the development process is divided into discrete steps, where each step is performed by specialized personell, and communication between steps is document oriented. How a team actually works, and how Scrum supports this will be explored.

1.2 The Ideal Spectator

This thesis is on a rapidly emerging part of software engineering. Hopefully, it has some value for the following audiences:

For readers with little or no previous knowledge of software development, some parts will probably probably be difficult to follow. However, it should contain enough basic theory to make most parts understandable.

Students of computer science or related subjects should get some insight into the day-to-day situation of the industry. Most parts could be of interest.

Practitioners interested in adopting Agile methods can possibly gain insight's valuable when eventually trying out Agile methods. This is part of the target audience.

Experienced Agile practitioners are probably aware of most parts presented. It should be read as a story, there might be concepts worthy attention.

Researcher interested in Agile methods should consider this thesis as an interesting addition to the rather scarce pool of other studies on Scrum. Absolutely in the target audience.

1.3 Structure And Naming

A quick word on the naming of the chapters is in order. As the term "Scrum" originates from rugby, the naming of chapters also adopts a rugby or sport metaphor viewing the project as a match.

Chapter 2: The Game: Rules and Tactics defines the rules of the game. It provides some background in order to explain the field of software engineering in general, and to highlight the research questions. A particular focus is on giving an introduction to Agile methods and Scrum.

Chapter 3: Seating arrangements is about the research design. As with sports, a spectators viewpoint of is important when regarding a later description of a game, by knowing whether the reviewer played the or listened to radio. This chapter outline the theoretical background and describe the research approach used.

Chapter 4: Match day: Live scores is the findings-section of this thesis which answers the initial research question. This chapter describes the case through three different but related aspects: contextual, chronological and the adoption of Scrum.

Chapter 5: Results discuss the findings, and addresses the two secondary research questions.

Chapter 6: Final table presents the final conclusions, possible improvements and further work.

Chapter 2

The Game: Rules And Tactics

The process of creating software is a complex undertaking. In some cases it is a technical challenge, but almost always when creating large systems there are several other important components such as communication and teamwork. Brooks describes the work of a software developer in the following way:

> The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.[Brooks, 1987, page 7]

This poetic description notwithstanding, creating good software is difficult. It has been plagued with several high profile failures, and undertaking large software development projects is often associated with some degree of uncertainty. A general rule when managing a software project is that of the often cited "Iron Triangle" shown in figure 2.1. There are 3 interdependent priorities in any project; cost, time and quality. Of these three, a project can arguably predict and control only two [Atkinson, 1999].

This chapter is about different approaches to software engineering, described in the next sections, is the systematic approach to creating software, a discipline commonly known as *Software Engineering*.

2.1 Software Engineering

Software engineering includes according to Sommerville [1995] specification, development, management and evolution of software systems. IEEE[1] gives

[1] Institute of Electrical and Electronics Engineers

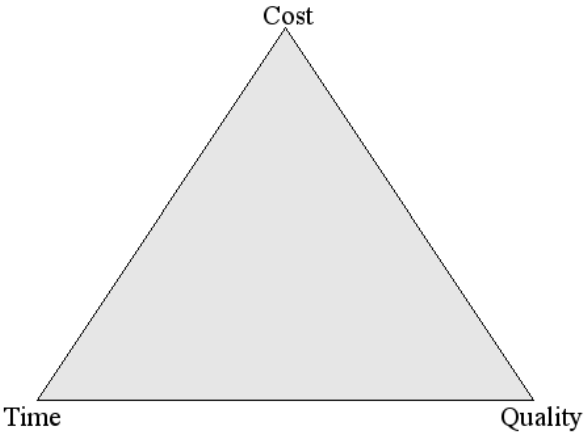


Figure 2.1: The iron triangle. From Atkinson [1999].

the following definition:

> **software engineering.** (1)The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1) [IEEE, 1990].

The above definition states that software engineering is about a rigorous approach to building software. The term, which originated at a NASA workshop in the late 1960s [Sommerville, 1995], imposes the image of rigour, care and assurance normally associated with engineering[2]. The level of rigour needed when creating software, and whether or not engineering is a appropriate metaphor for software is somewhat debated (Bryant [2000] argues that the engineering metaphor does not fit software and that software should be grown, rather than built). This notwithstanding, Software engineering is about solving the problems inherent in software development, with the goal of making it as cost-effective, predictable and successfull as possible.

From the definition given by IEEE of Software Engineering, it has three components: development, operation and maintenance. Although these three can be intertwined, this thesis is mainly about the development component. Again, the definition by IEEE is useful:

> **software development process.** The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, trans-

[2] *The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, ...*[eng, 1941]

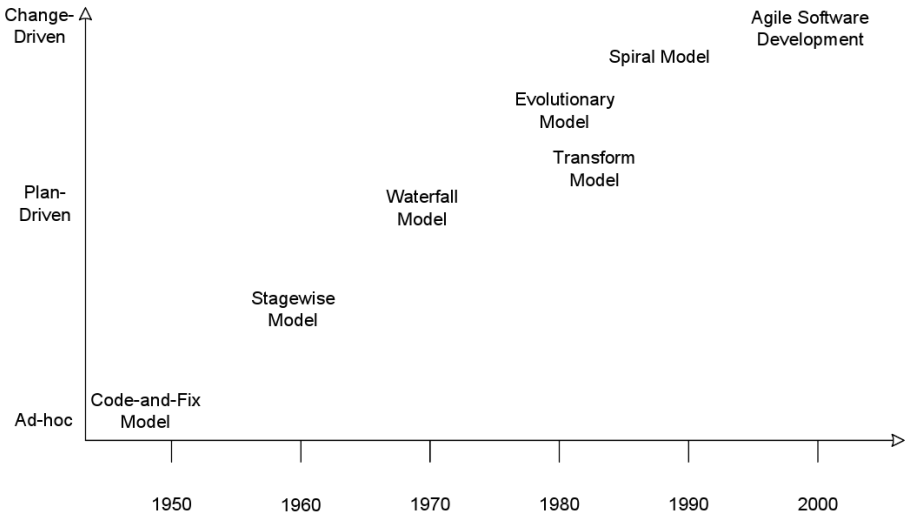


Figure 2.2: Evolution of Software process models. From Salo [2007].

> forming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. Note: These activities may overlap or be performed iteratively. See also: **incremental development; rapid prototyping; spiral model; waterfall model**. [IEEE, 1990]

Software development as defined above is about the systematic approach to software development. The following sections will give a brief outline on how the discipline has evolved over time, and further present some methods common in software engineering.

2.1.1 A Brief History Of Methods

As previously described, software engineering as a discipline emerged in the 1960s, and has in the following years grown in scale and importance. The various models used in software engineering have as figure 2.2 shows also evolved. While not all of the terms in this figure are explained, especially two of them are central in the later chapters; waterfall models and Agile Software Development.

According to Avison and Fitzgerald [2003] software engineering has gone through 4 different eras; pre-methodology, early methodology, methodology and are now in the post-methodology era.

Pre-methodology era was the early years of pioneering in the 1950s and 1960s. The emphasis was on programming and problem solving, often coupled tightly with hardware constraints. There was little emphasis on user needs, problems were usually very technical in nature. Development was done without using formalized processes, and the effort was usually individualistic, with simple project management and control [Avison and Fitzgerald, 2003].

Early Methodology era in the late 1970s and early 1980s focused identification of phases and stages. The aim was to improve the management of software development, and to use specific techniques in the different phases. This era introduced discipline into the process [Avison and Fitzgerald, 2003], but had several issues. Among others were difficulty of meeting business needs, dissatisfied users and inflexibility.

Methodology era was the answer to the issues from the earlier methodologies, and in the 1980s and early 1990s several new approaches emerged. These specified a recommended collection of phases, rules, techniques, tools, documentation, management principles, and training. Although the recommendations followed several different approaches, the motivations were the same; To achieve better end products, to improve the development process, and to standardize on methodology.

Post-Methodology era can be considered as the reappraisal of some of the values from the pre-methodology era. This is attributed to the use, misuse, or nonuse of the elaborate and heavy methodologies of the 1980 and 1990s [Avison and Fitzgerald, 2003], which generally was rejected by practitioners. Reasons for this rejection include disappointing productivity, overly complex and unrealistic sets of rules, and the requirement to use costly and complex tools. The reappraisal included among other approaches such as increased tool use or external development, incremental methods which is one of the cornerstones of Agile software development [Avison and Fitzgerald, 2003].

The methodologies from the two methodology eras are from now on broadly labeled as traditional methods. The following section elaborates on the use of this label.

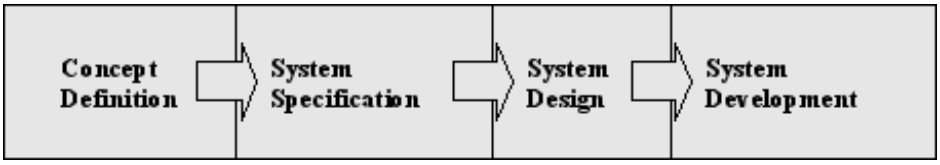


Figure 2.3: The main steps of traditional software engineering

2.2 Traditional Methodologies

Defining traditional software development is difficult. According to Boehm, traditional methods include extensive planning, codified processes, and rigorous reuse in order to create software efficiently and predictably. They are in other words essentially plan-driven. With planning Boehm means:

> ... documented process procedures that involve tasks and *milestone plans*, and product development strategies that involve requirements, designs, and *architectural plans* [Boehm, 2002]

In the above quote three expressions are central to understanding the distinction between Agile methods and traditional methods; *requirements, design and architectural plans*. Compared with figure 2.3, requirements encompass step 1 and 2, while designs and architectural plans are parts of both step 2 and 3.

As figure 2.3 shows, the first step is to outline the concept of the system, creating a broad statement of the users requirements. Setting the projects bounds and the direction for the whole project [Hawryszkiewycz, 2001].

Requirements are the second step of traditional methods. This is usually the process of creating a (extensive) requirement document which specify the required functionality, expected business improvements and detailed objectives of a software system. These documents are created by specialized analysts using specialized techniques such as object oriented techniques or structured systems analysis [Hawryszkiewycz, 2001]. When finished, the requirements document is essentially frozen, and is rarely changed.

Design is the third step where the requirement document are used to design and create plans for the systems architecture. First a broad architectural design specifying logically how a system will fullfill the requirements is created. Then a physical design is created based on the architectural design, which in detail explains how the logical design should be developed [Hawryszkiewycz, 2001].

Another take on the traditional way of doing things is in "Software Process Models" by Sommerville. In this paper, the distinction is between *Specification-based models* and *Evolutionary development models*. Specification based models relies on a set of specifications that are frozen and is the basis of further development. Sommerville writes that such models are most applicable to large system-engineering projects where the need for predictability is high.

Plan driven or specification based models can be compared with the process of building a house or a bridge. Get the basic requirements from the buyer and the surroundings. Create the architecture sketches and plan the building process. Build according to the architecture and progress plans. Deliver on a date set in the contract. How the build is done is of small importance, and requirements are difficult to change in the middle of the build. This basic outline of the process is applied to software

To further illustrate the traditional approach, the following section will elaborate on some common traditional methods.

2.2.1 Traditional Methods

The prime example of traditional methods, often used as the opposite of Agile, is the linear waterfall model. Also the later, spiral model and Rational Unified Process are labeled as traditional models, and is used as examples of the more modern breeds of traditional methodologies.

The following section elaborates on these three methods.

Waterfall And Linear Methods

The archetypal software development model is the linear or waterfall model[3]. It is quite similar to the house-building analogy from the last section.

The waterfall model was initially proposed in Royce [1987] in 1970, and is a true product of the early-methodology era. Royce's original model has 7 consecutive steps, illustrated in figure 2.4. The model in figure 2.4 is a strictly linear process where each step finishes completely before the next step begins. The analysis phase requires a finished set of specifications, and the design phase depends on a finished analysis. Each phase produces a range of documents, which is used in the next phase. Royce's paper also has versions of this model where there are some feedback between consecutive phases, although with the same basic process. Feedback can be done in

[3] The model is popularly known as the Waterfall model, because the products of each phase cascade from one to another creating a "waterfall" [Sommerville, 1995]

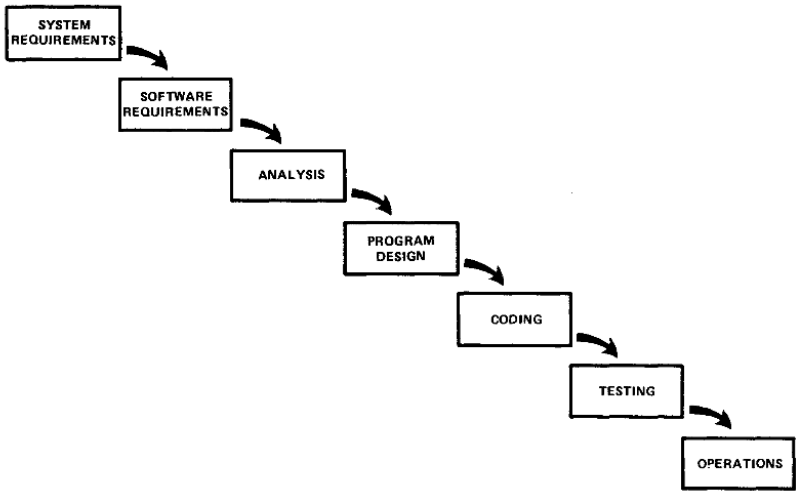


Figure 2.4: Royce's initial lifecycle model, now popularly known as the Waterfall model. From Royce [1987]

several ways. According to Sommerville [1995] one example is when the experience from the last phase (operation) can be used as input in any of the other phases. It is however usual to have a single sign-off point where one activity terminate, shifting focus to the next [Hawryszkiewycz, 2001].

Structured teams are often used in linear projects, where team members are assigned to specific phases [Hawryszkiewycz, 2001]. System analysts produces an analysis document and passes this on to the architectural team. The architects then produces a system design document, which is the basis of the development phase carried out by a development team. The development team produces a system which is tested by testers, and lastly implemented. It is possible that only management is a part of the entire process, making documentation the essential information-carrier from one phase to the next. Quality assurance is ensured by validation of the produced documents at the end of the phase.

Despite being criticized from the inception [Royce, 1987], the waterfall model is still in widespread use. It has been adopted as a general standard in many software procurement processes, and is by this still an important model in software engineering [Sommerville, 1995].

The Spiral Model

The spiral model is a generic model proposed by Boehm in 1986 based on experiences from TRW [Boehm, 1986]. It is an example of the methodology-

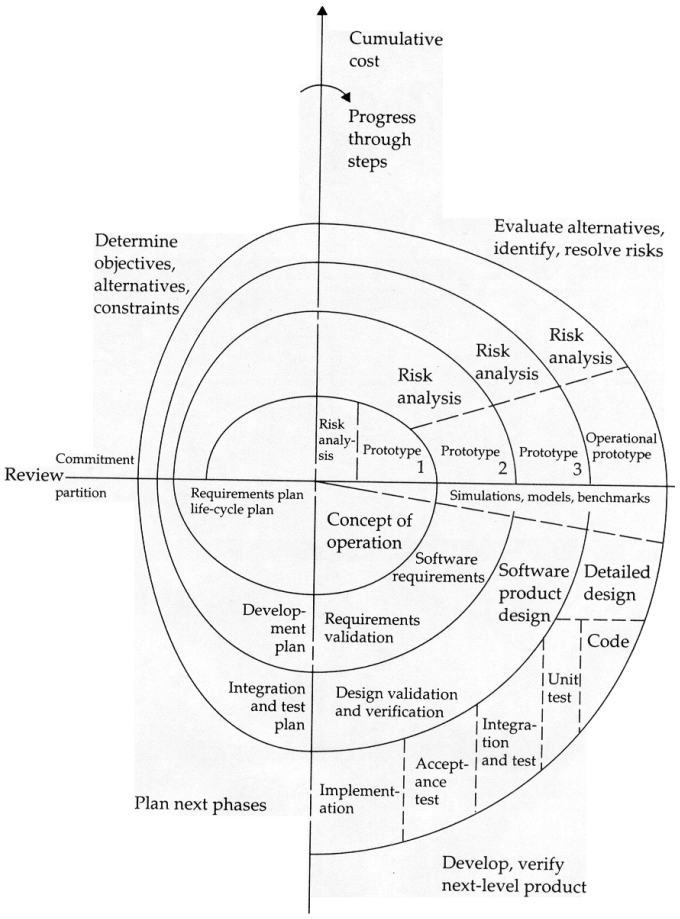


Figure 2.5: The spiral model. From Boehm [1986]

era. The focus is on recognizing and handling risks in the development process. As the name hints at, the spiral model takes the form of a spiral where each loop in the spiral represents a phase of development. Each loop has a different emphasis, the first can be about system feasibility, the next with requirements and so on. Each phase or cycle goes through some planning and risk analysis as well as prototyping of the system. As this is a generic model it is up to the management to decide the focus of each cycle.

The spiral model can be used as a encapsulation of other models. For instance one phase can use prototyping to explore and specify the system, while the next can be a waterfall-like phase where the prototype is developed into a full scale product. The main points are that the development is done in cycles which support each other, and that risks are evaluated before starting a new cycle. These cycles are quite long (2-3 months to a year or more) and risks are evaluated through surveys, interviews and benchmarking tools. It

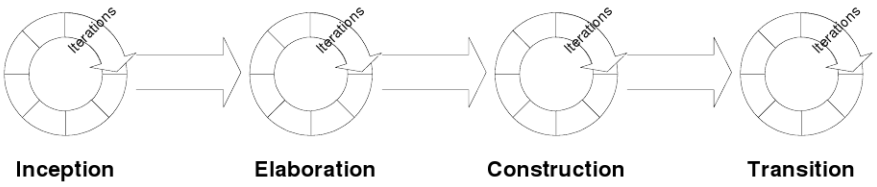


Figure 2.6: Rational Unified Process. From Abrahamsson et al. [2002].

is according to Boehm less document driven than the waterfall model, and is most suitable for large complex systems. Cohen et al. [2004] notes that while the spiral model is more adaptable than the waterfall model, it still relies on long cycles and have a heavy component of analysis and planning.

Rational Unified Process

Rational Unified Process, abbreviated as RUP, was developed to complement UML[4] It is an iterative approach for developing object-oriented systems [Abrahamsson et al., 2002].

As figure 2.6 shows, RUP consists of four phases named inception, elaboration, construction and transition, which can roughly be mapped to the phases shown in figure 2.3. The phases are further divided into several cycles or iterations. Further, RUP specifies nine workflows ongoing through the entire project, and divides responsibilities among 30 different roles, such as architect, designer, design reviewer or configuration manager). RUP also have an emphasis on tool automation and modelling tools [Abrahamsson et al., 2002].

Labelling RUP as a traditional methodology can be argued and is labeled as an Agile method in "Agile software development methods: Review and analysis" [Abrahamsson et al., 2002] labeled as an Agile method. The reason for including RUP in this context is because of the number of different roles, practices and techniques, as well as the reliance on tools, documentation, planning and architecture. RUP is an example of a heavy methodology from the late 1990s so complex that it requires significant training just to know the basics. Abrahamsson et al. [2002] writes that RUP is supposed to be tailored to specific projects, but the number of practices means that it fails to provide clear implementation guidelines.

[4] Unified Modelling Language is a standard of notations to represent object models [Hawryszkiewycz, 2001], which according to Avison and Fitzgerald [2003] is one part of the methodological era.

2.2.2 Issues

Traditional methods, as described so far, spans over a large range of methods from the somewhat archaic waterfall to (arguably) RUP. Their common characteristics are mainly in the planning aspects and an emphasis on specifications, architecture and design. As section 2.1 illustrated, the methods employed have evolved considerably

There are a number of issues associated with these traditional methodologies.

The linear methods has, as previously stated been criticized from the inception, also by Royce in the paper initially containing the waterfall model:

> I believe in this concept, but the implementation described above is risky and invites failure Royce [1987]

The argument was that the process lacked feedback from one stage to another [Sommerville, 1996]. In the same paper, Royce proposed a modification with feedback loops, but these have largely remained forgotten.

In Brooks [1987], the extensive planning done in the specification and design stages of traditional methods is criticized. Brooks argues that the hard parts of creating software is not to develop the system. Problems in this phase are not that important, compared with conceptual errors done while creating the specification and designing the system. Further, he argues that these issues cannot be resolved without fundamental revision in the industry that provides for iterative development and specification.

Traditional methodologies focuses on making software development into a repeatable, defined and predictable process [Cohen et al., 2004]. This aim has been criticized, because when the problem is complex, changing, or not completely understood, an inflexible, large methodology is especially unsuitable.

Agile methods, elaborated on in the next section, are designed to address some of these issues.

2.3 Agile Software Development

As mentioned in section 2.1.1, Agile software development was one of the approaches adopted in the post-methodology era.

A good, concise definition of what Agile means in software engineering world is difficult to come by. However, the definition of the word Agile is a start:

> S: (adj) agile, nimble, quick, spry (moving quickly and lightly) "sleek and agile as a gymnast"; "as nimble as a deer"; "nimble

> fingers"; "quick of foot"; "the old dog was so spry it was halfway up the stairs before we could stop it"
>
> S: (adj) agile, nimble (mentally quick) "an agile mind"; "nimble wits" WordNet

These two definitions give some insight into the general idea of Agile methods. Instead weighing down a process with too much "fat", Agile methods aims to be lean, adaptable and intelligent.

Agile methods are inspired by by Lean manufacturing, which was a trend started in Japan after world war II. Lean manufacturing is about managing without large stores of supplies, producing only according to demand, and empowering workers to do their job as well as possible [Cohen et al., 2004].

Agile methods are the principles of Lean manufacturing applied to software engineering. Agile encompasses several practical approaches which is perhaps best summarized by the guiding principles of Agile Software Development stated in the Agile Manifesto:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more. Beck et al. [2001]

These statements form the core of Agile methods. The focus is on doing the things directly related to the software development with rich communication[5] between all interested parties, offering an alternative to the documentation driven, heavyweight software development processes [Cohen et al., 2004].

In recent years, Agile methods, and especially XP, have attracted substantial interest in academia, including several literature reviews [Abrahamsson et al., 2002, Cohen et al., 2004, Dybå and Dingsøyr, 2008]. Studies report that the methods are easy to adopt and work well, especially in small, uncritical projects with changing requirements. Improvements have been found in the following areas: customer collaboration, work processes for handling defects,

[5] Richness is connected to the modes of communication available. Face to face communication using a drawing whiteboard is considered as very rich, while paper is considered as the opposite [Cockburn, 2002, figure 3.14].

learning in pair programming, thinking ahead for management, focusing on current work for engineers and estimation [Dybå and Dingsøyr, 2008].

2.3.1 Motivation And Background

Traditional software development methods as have several issues as described in 2.2. The actual use of these methodologies are described in Abrahamsson et al. [2002] to be more symbolic than anything else, and something that is viewed with skepticism by many industrial software developers. Agile methods can be regarded as the practitioners response with more useable practices.

In section 2.1.1, the post-methodology phase was described as the reappraisal of values from earlier years. Some of the roots of Agile methods are described in "Iterative and Incremental Development: A Brief History"[Larman and Basili, 2003]. This article has several good examples of early use of Agile-like methods:

NASAs project Mercury used in the early 1960s test-first development very much like XP (see section 2.3.2).

Iterative and incremental methods (IID) was further used in various defense programs in the 70s, including the Light Airborne Multipurpose System which was a 200-person-years project on helicopter to ship weapon systems. This project used one month iterations like Scrum (see section 2.5), and delivered the system in 45 iterations. NASA continued to use similar methods, with the Space Shuttle project as one high profile example [Larman and Basili, 2003].

This type of methods was used and discussed significantly through the 80s and 90s. Waterfall-influenced processed still remained as the most used and referred to. This is attributed by Larman and Basili [2003] to the simplicity of the model, the orderly process with simple and understandable milestones, and the continued promotion in texts, courses and consulting organizations which labeled it as the idealized process.

In February 2001, a group of practitioners created the Agile manifesto [Beck et al., 2001] which "officially" started Agile software development. The term has since then become an established part of software engineering.

2.3.2 Agile Methods

The following sections contains an overview of a selection of existing Agile approaches. Scrum has not been included in these, as it is the main fo-

cus of this thesis, and consequently warrants the somewhat more thorough explanation given in section 2.5.

Extreme Programming

Extreme programming or XP is the most widely researched Agile method [Dybå and Dingsøyr, 2008]. It evolved from problems with the traditional development models, and is based on practices that had been found effective in software development in the earlier decades. Ideas from among others Scrum, evolutionary development and the spiral model is incorporated into XP. It is essentially a theorized version of common sense principles and practices taken to extreme levels, giving its name. There also exists a newer, more extensive version, XP2, but the traditional XP described in Abrahamsson et al. [2002] is covered here.

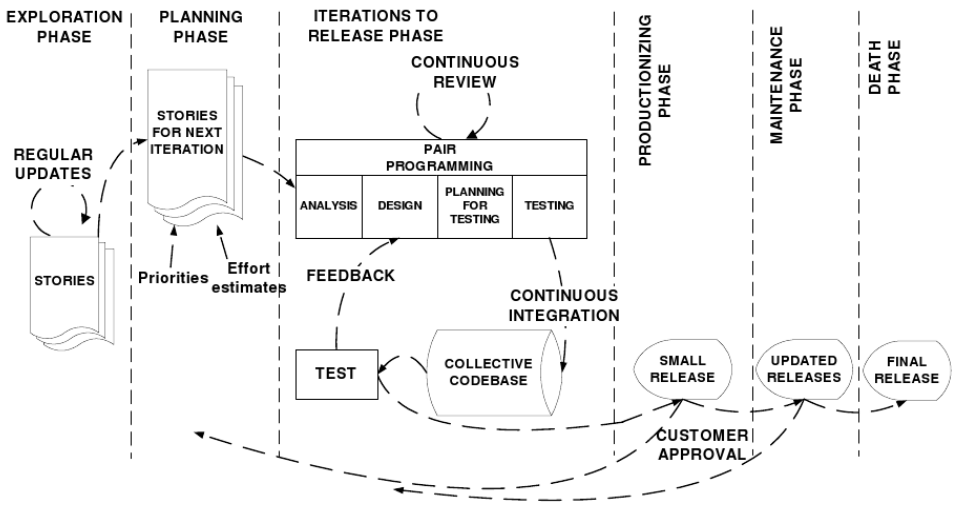


Figure 2.7: The XP Life cycle. From Abrahamsson et al. [2002]

As XP is based on experience with existing methodologies, it gives quite clear advice on how a software project should be run. XP describe the distinct phases of a project, roles and responsibilities, and 13 concrete practices:

- Planning game – Planning done by cooperation between programmers and customer; programmers estimate effort, the customer decides on scope and timing.
- Small/short releases – New versions of the system is released rapidly, at least monthly.

- Metaphor – A "shared story" describing how the system works, guiding development.
- Simple design – Design the simplest possible solution which is possible at the moment.
- Testing – Development is driven by tests, which are implemented before the actual code. Tests are run continuously.
- Re-factoring – Restructuring the system by improving and cleaning up the code whenever possible.
- Pair programming – Two people write code using one computer.
- Collective ownership – Code can be change by anyone, anytime.
- Continuous integration – Code is integrated into the code-base as soon as it is ready, tested and built.
- 40-hour week – Team members is not allowed to work more than 40 hours a week. More than two overtime weeks is treated as a problem.
- On-site customer – The team has access to a customer at all times.
- Coding standards – Programmers follow coding rules, with emphasis on communication through code.
- Open workspace – The team works together in a open workspace optimized for cooperation.
- Just rules – Team creates and follows their own rules.

XP is mainly suited for small and medium sized teams (between 3 and 20 members), working with technology which supports "graceful change". As some of the practices in XP requires discipline, it is important not to have members or management which resists these [Beck, Oct 1999].

When regarded as a whole, adopting all the practices of XP can be a difficult task. Both the creators and experience reports agrees on this, and suggests that XP should be adopted gradually, and tailored to the needs of the individual projects. Beck writes:

> If you want to try XP, for goodness sake don't try to swallow it all at once. Pick the worst problem in your current process and try solving it the XP way.

In addition to experience reports, there have been a substantial amount of studies on XP, dominating research on Agile methods [Dybå and Dingsøyr, 2008]. Conclusions are mostly positive, with documented gains in productivity [Abrahamsson et al., 2002].

As figure 2.7 suggests, XP specifies most aspects of a development process. It is especially strong on the engineering aspects through the 13 practices. XP has often been combined with other methodologies such as Scrum.

The Chrystal Methodologies

The Chrystal family is a range of methodologies designed for projects with different requirements. The choice of methodology is based on two variables; criticality of the system and size of the project, which decide which methodology is appropriate. Each member of the family is marked with a color, and the darker the color the heavier the methodology. Crystal clear is most suitable for small simple projects, while Crystal red is for the large, complex ones with the other colors for the intermediate spectrum.

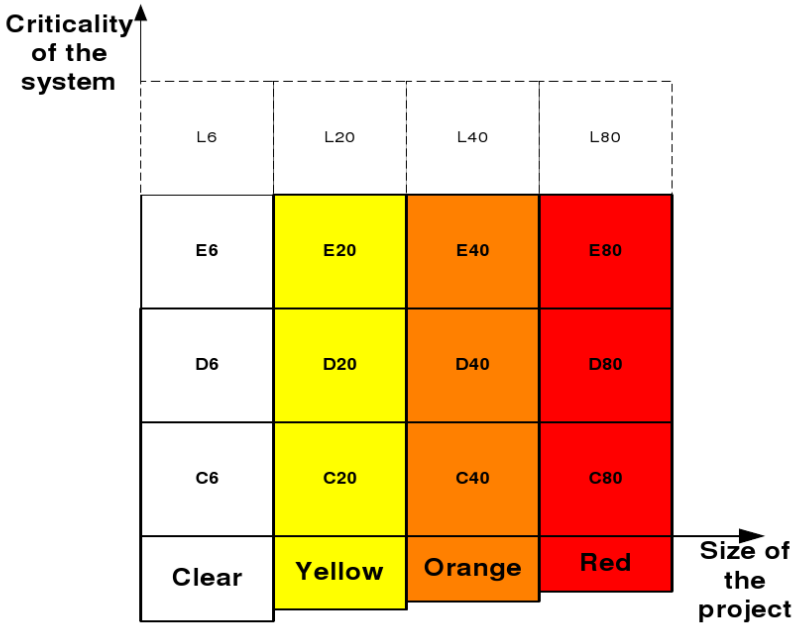


Figure 2.8: Dimensions of the Crystal methodologies. From Abrahamsson et al. [2002].

The crystal family have some common principles:

- The number of intermediate work products are reduced as the team produce more running code, and communicate better.
- Conventions in a project is shaped and evolves over time.

- The amount of overlapping work, and information holders is determined by the shifting bottlenecks in the system.

Two rules are common:

- Incremental development, with increments between 1 and 3 months, with a maximum of 4.
- Pre- and post-increment reflection workshops, and with a preference to hold one in mid-increment.

Further there are two base techniques:

- Tuning the base methodology to fit the project based on project interviews and workshops.
- The reflection workshop technique.

Crystal Clear and Crystal Orange are the only members of the family which actually have been specified in detail.

Crystal clear is designed for very small projects with up to six developers, and is tolerant and free of ceremony. It describes some roles, policies and work products, but does not require any documentation to be created [Abrahamsson et al., 2002].

The following elements are regarded by Cockburn as important success factors in Crystal Clear; focus on close seating and close communication, frequent deliveries, information from real users and use code versioning tools. Further he summarizes the method in the following statement:

> I don't think you can get any sloppier than Crystal and still plan on having better-than-even odds of completing successfully. [Cockburn, 2002]

Crystal Clear require the following tools: compiler, versioning and configuration-management tool and printing whiteboards for documentation.

Crystal Orange is for medium sized projects with 10 to 40 project members. Compared to Crystal clear, more roles, work products and policies are suggested, and in addition the members of the project are divided into functional teams such as system planning and project monitoring. These teams are then split into cross-functional groups. Crystal orange requires the following tools in addition to the tools of Crystal clear: versioning, programming, testing, communication, project tracking, drawing, and performance measuring.

There have been no published studies on Crystal methodologies [Abrahamsson et al., 2002, Dybå and Dingsøyr, 2008]. However, the proposed scope of these methodologies is an example of an Agile method which can scale be-

yond the commonly cited "home grounds" of small, non-critical and volatile internet systems.

2.4 Comparison: Agile VS. Traditional

Before elaborating Scrum, it is fitting to review and compare the fundamental values of the traditional and Agile approaches.

In section 2.2 traditional approaches were described as plan-driven. The planning spectrum in figure 2.9 compares the level of planning in different approaches. Traditional methodologies are generally to the right in this spectrum.

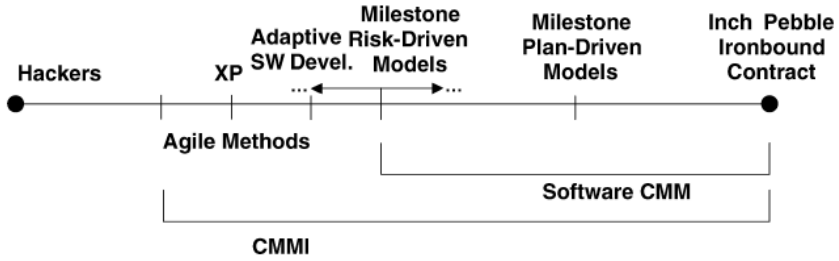


Figure 2.9: The planning spectrum. Unplanned hacking is on the far left side of the spectrum, while micromanaged planned approaches are on the extreme right. From Boehm et al. [2002].

A more detailed comparison is in table 2.1. It is outside of the scope of this thesis to explain all of the terms in this comparison. However, the following aspects and the differences are the important ones when relating this comparison with the research questions in this study; control, knowledge management, communication and assignment of roles, and especially fundamental assumptions.

Agile methods puts people in control, in contrast to traditional methods which relies on pre-described processes and. This relates to RQ1.

Agile methods aims to be light and easy to implement, supporting practice with concrete guidelines. Reports suggest easy adoption. Traditional methods preaches predictability by meticulous specification of the process, and is rarely implemented as described [Cohen et al., 2004]. This is the central point of RQ1.

	Traditional	**Agile**
Fundamental assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management Style	Command-and-control	Leadership-and-collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual – favors specialization	Self-organizing teams – encourages role interchangeability
Communication	Formal	Informal
Customer's Role	Important	Critical
Project Cycle	Guided by tasks or activities	Guided by product features
Development Model	Life cycle model (Waterfall, Spiral, or some variations)	The evolutionary-delivery model
Desired Organizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors object-oriented technology

Table 2.1: Comparison between Traditional and Agile software development. From Nerur et al. [2005].

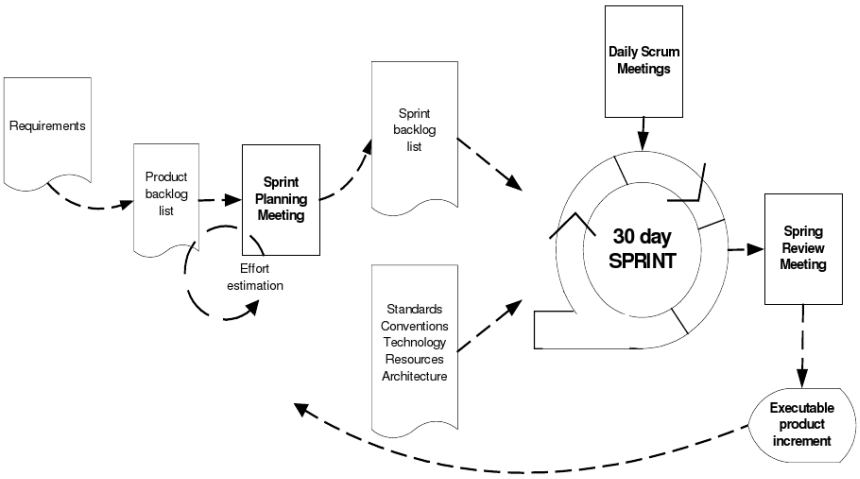


Figure 2.10: The Scrum Process. From Abrahamsson et al. [2002]

Agile methods focus on good communication and tacit knowledge, instead of formal communication through documents and explicit knowledge[6]. Both RQ1 and RQ2 are influenced by this difference.

Agile methods favors self organizing, organic teams where the collective is more important than the individuals. Traditional methods favors specialization, where individuals are assigned to distinct tasks. This relates to teamwork, and is directly relevant to RQ2.

2.5 Scrum

[htb] The term Scrum originates from rugby[7], where it is a special strategy for getting an out-of-play ball back into play [Schwaber and Beedle, 2001]. In engineering, the term Scrum was first used in Takeuchi and Nonaka [1986] where they presented an adaptive, quick, self-organizing product development process [Abrahamsson et al., 2002]. This approach was primarily used in non-software products at Honda, Canon and Fujitsu.

Inspired by this Jeff Sutherland and Ken Schwaber applied this method to software development at Easel Corp. in what would eventually be known as Scrum. Scrum was later refined in writing by Beedle et al. [2000] in

[6] Tacit knowledge encompasses unexpressed internal knowledge, while explicit knowledge is expressed through writing [Walsham, 2001].

[7] It is worth noting that the rugby heritage means that Scrum is not an acronym, and should not be uppercased

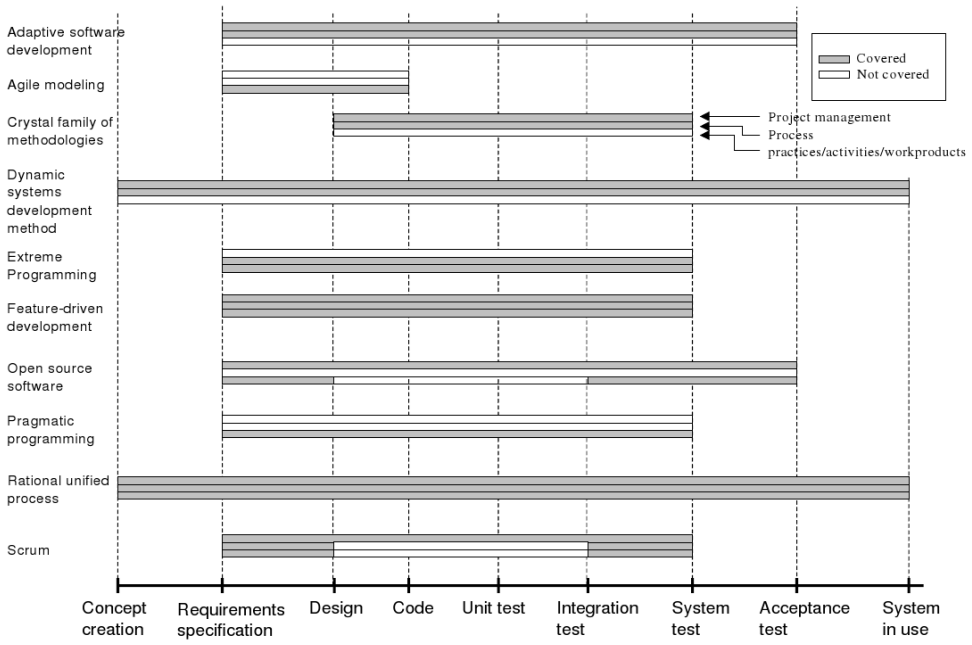


Figure 2.11: Life-cycle support of various Agile methods. From Abrahamsson et al. [2002]

"SCRUM: An Extension pattern Language for Hyperproductive Software Development", and has since then accumulated much related literature.

Scrum is mainly an agile approach to management of software development projects [Abrahamsson et al., 2002] and it does not specify details about how the development is done. Scrum is designed to encapsulate the existing practices. Management of the process is empirical, inspired by industrial process control theory, where the process is controlled by constant monitoring of progress and doing adjustments based on this knowledge [Schwaber and Beedle, 2001]. Figure 2.11 illustrates this, and compares Scrum with a number of other methods. Compared with XP, Scrum supports the project management aspects of a project, but has little support for the more detailed aspects of development.

2.5.1 The Fundamentals

As figure 2.10 shows, the core practice in Scrum, described by Schwaber as the skeleton, is the Sprint. This is a short iteration, typically 30 days, of the development process where a *potentially shippable* increment of the system is developed and delivered. A potentially shippable increment is defined by Schwaber as tested, well structured code, compiled into an executable with

the necessary documentation. Any number of sprints may be performed before finishing a project.

While a sprint and the deliveries may be the skeleton of Scrum, the heart is in the different management practices [Schwaber]. These include the use of various artifacts, discussions in a number of meetings, and responsibilities divided among a small number of roles.

Artifacts

The following artifacts are the fundamental ones;

The Product Backlog is simply a prioritized queue of features wanted, or work to be done on the system. Items can be added to the product backlog at any time in the project and it can be re-prioritized. It is considered to be a dynamic document [Schwaber and Beedle, 2001], very unlike the specification documents described in section 2.2.1. The effort needed to realize the product backlog items is estimated prior to implementation, but these estimates are not supposed to be binding[Schwaber].

Sprint Backlog is in essence a list of tasks to be done in a sprint. Before starting a sprint, the team selects an appropriate number of items from the prioritized product backlog which together form a sprint increment. These items are then decomposed into concrete tasks needed to create the delivery. Together, these tasks form the sprint backlog which is stable through a sprint [Abrahamsson et al., 2002].

Burndown Chart is not a part of figure 2.10, but is still a recognizable and often referred to part of Scrum. Schwaber describes it as a chart showing the remaining work in a sprint as time progresses. It is useful as a tool to indicate when a sprint backlog is finished, and realizing if adjustment of the effort is needed. Figure 2.12 show an example.

Meetings

The artifacts discussed in the previous section are closely linked to these meetings:

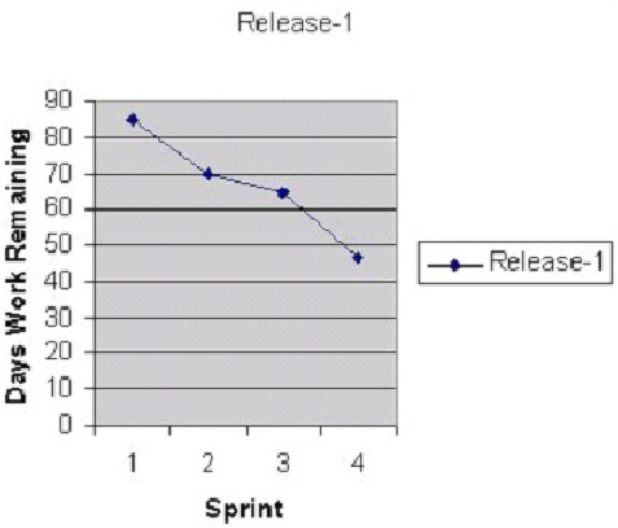


Figure 2.12: Sample burndown chart. From Schwaber and Beedle [2001].

Sprint Planning Meeting is a done prior to a sprint, and is used to create the sprint backlog. The objective of this meeting is to plan the the next sprints product increment. The input is the prioritized product backlog, along with descriptions from the product owner. This is then weighed by the team together with knowledge of previous deliveries and past performance [Schwaber and Beedle, 2001]. The result is a commitment from the team to release a part of the system, and a relatively detailed plan to fullfill this delivery.

Scrum Meetings are short, 15 minute meetings, usually done daily aimed to ensure communication and establish the current progress and identify impediments [Schwaber and Beedle, 2001]. The meetings core can be summarized by three common questions:

1. What have you worked on since last the last daily Scrum?
2. Does there exist impediments which hinders your work?
3. What will you work on until next daily Scrum

It is usual to do these meetings in a pre-designated place, not necessarily a meeting room, but one readily accessible to the team [Schwaber and Beedle, 2001].

Sprint Review Meetings are performed at the end of a sprint, and is where a product increment is demonstrated by the team for the stakeholders

to get input on the progress, and further development. It is according to Schwaber and Beedle [2001] usually 4 hours long.

Retrospective Meetings are held by the team when appropriate, and is an opportunity to look back and identify possible improvements to the process. The other meetings are geared mainly towards the product increments, while the retrospective meeting is used to reflect and improve on the process as a whole Schwaber.

Roles

Agile methods preach people before process. People in a Scrum project is divided into three different roles; the product owner, the Scrum master and the team:

Scum Master is responsible for the success of Scrum [Schwaber and Beedle, 2001], but does not manage the project. The Scrum master instead works with the team in defining and realizing the goal of the product. While some aspects of the Scrum master role is project management related, this role is described as more like a coach ensuring that the other actors follows the rules laid out by Scrum [Schwaber].

A Scrum Team is the main part of any Scrum project, and is responsible for designing, developing, testing and in many cases, deploying the software product. A Scrum team is self managing, self organizing and cross functional [Schwaber]. It should, according to Schwaber and Beedle, include people with the necessary skills to achieve the team goal. Developers, testers, architects, analysts, designers and so on. The ideal team is [Schwaber and Beedle, 2001] seven people, plus minus two.

The Product Owner is the link between the scrum team and the customers and stakeholders of the project. Stakeholders can be management, users and IT management personell, basically everyone interested in the final outcome of the system. The product owner manages the product backlog on behalf of the stakeholders. According to Schwaber and Beedle there should ideally be only one product owner.

Phases In a Scrum Project

Scrum has three main phases; *pre-game*, *development* and *post-game*.

The Pre-Game phase consists of planning and design. The planning consists of compiling a initial product backlog. Some high level design can then be done based on the product backlog. Last in this phase, preliminary plans for the releases are prepared [Abrahamsson et al., 2002]. The pre-game phase should be as short as possible [Schwaber and Beedle, 2001].

The Development phase, or game phase, is the agile part of Scrum where the team actually develop the product. In the development phase the scrum team is responsible. This phase is treated as a black-box divided into a number of sprints, where the team is protected from changes. Changes such as requirements, time-frame, quality, resources, release plans and specifications are handled only between sprints [Abrahamsson et al., 2002], while product increments are delivered steadily. This makes Scrum flexible, ensuring that developers are not constantly interrupted, while focusing on frequent and predictable deliveries.

The Post-Game phase is when the release is closed. This phase is reached when the requirements are completed, quality demands are met, business dictates termination or other reasons for closing the project become evident. In this phase necessary and remaining system testing, documentation and integration is performed [Abrahamsson et al., 2002].

To highlight, I will illustrate a model Scrum project:

A need for a new scoring system is identified by the President of the Norwegian Rugby Union. They decide to buy it, and hire a consulting company which have good experiences with Scrum. The consulting company, promptly identifies the team and Scrum master while the Union finds a product owner which knows what is needed.

The team and product owner then go through a short (1 week) pre-game phase and creates an initial product backlog. They then hold a sprint planning meeting, creates the first sprint backlog and starts sprint 1. On conclusion of Sprint 1 they present the increment of functionality created, receives an updated and prioritized product backlog, create the next sprint backlog and starts sprint 2. After sprint 2, the system has baseline functionality and is put to use as is.

In the following months, the team finishes 4 more sprints. After sprint 6, the Rugby union management is satisfied and declares that the System has all the necessary functions and meets quality demands. A short post-game phase is carried through where the team polishes the user manual. The new system is then put to use and the rugby union from now on has complete control of all scored points.

2.5.2 Studies

As noted in the motivation for the study, the number of Scientific studies on Scrum is relatively low, with a high number of lessons-learned article. The studies presented briefly in this section are scientific studies of Scrum implementation. Summarized, their conclusions are that Scrum works well, especially when combined with other Agile techniques. However, with regard to teamwork it is has some limitations. Reported benefits are reduced overtime, fewer defects, increased customer satisfaction, and deliveries ahead of schedule.

In Mann and Maurer [2005], the researchers conducted a 2 year longitudinal case-study of Scrum in an industrial case. This organization introduced Scrum, and some other Agile practices (pair programming, unit testing and continuous integration). Their findings indicated that introducing Scrum decreased the use of overtime, and enabled a more sustainable pace for developers. In addition, the results showed a tendency towards increased customer satisfaction.

"Customizing agile methods to software practices at Intel Shannon" [Fitzgerald et al., 2006] is a 3 year exploratory case study of the use of XP and Scrum. At Intel, a mix of XP and Scrum was used, where the development department tailored their own process based on the available practices from these two methodologies. Only 6 of the practices from XP was adopted, while notable modifications to Scrum included splitting large tasks across sprints, and using a two-stage planning process instead of one. The study found that the a la carte combination of practices worked very well, and led to commited usage. The reported benefits of the Agile methods included reductions in code defect density by a factor of 7 as well as delivery ahead of schedule in projects of 6-month and 1-year duration.

Moe and Dingsøyr [2008] is a ethnographic study on Scrum and how it supports teamwork. The findings are structured along the "Big Five" from Salas et al. [2005]. The results were that Scrum does not address the leadership function sufficiently, lacking clear advice on how to actually implement this. Backup-behavior and mutual trust are also issues, which might be resolved using other Agile techniques such as pair-programming.

Two master thesises on Scrum are worth to mention. Lervåg [2006] is a interpretive study of a Norwegian Scrum project, in some ways similar to this, but with a broader scope and employing different research methods. Tøsse [2007] is also a case study on Scrum, but one from another viewpoint, namely interdisciplinary culture studies. In this study, the main data was from interviews.

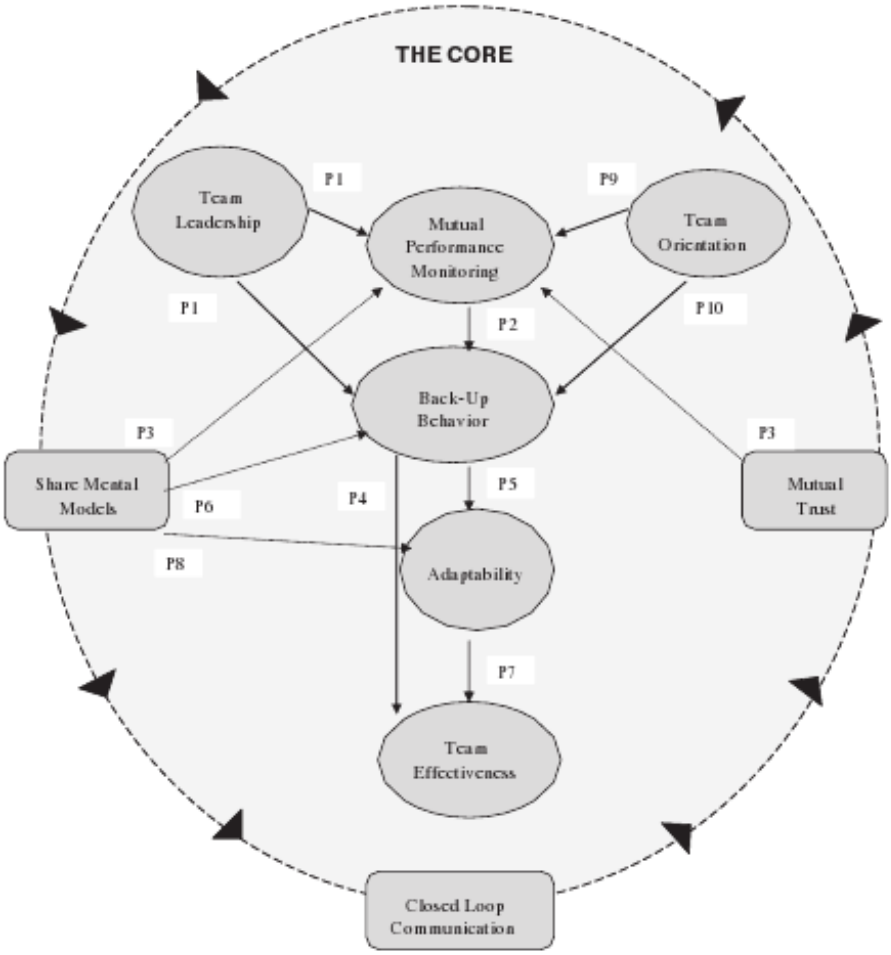


Figure 2.13: Relationships between the big five and coordination mechanisms. From Salas et al. [2005].

2.5.3 Scrum And The Big Five

Teamwork in Scrum is regarded as a black box. While the responsibilities are divided between roles, the personalities of the people involved are probably just as important as these. In the previous section, one study on Scrum [Moe and Dingsøyr, 2008] used the framework in Salas et al. [2005] to discuss teamwork in a Scrum project. The following section will briefly compare theory on Scrum with this framework.

Coordinating Mechanisms

Shared Mental Model

> Definition [Salas et al., 2005]: An organizing knowledge structure of the relationships among the task the team is engaged in and how the team members will interact.

Creating and maintaining a shared mental model is important in order to understand the task at hand to create a basis for cooperation. A shared understanding of team members priorities and work situation is also important in this mechanism [Salas et al., 2005]. Shared mental models directly affects mutual performance monitoring, back-up behavior and adaptability.

This aspect is according to Moe and Dingsøyr [2008] well supported by Scrum. Involvement by the product owner, planning through backlogs, review and retrospective meetings and the daily Scrum meeting are important aspects. Co-location of the team is also vital.

Closed Loop Communication

> Definition [Salas et al., 2005]: The exchange of information between a sender and a receiver irrespective of the medium.

In Scrum, there are essentially four two-way communication links. Developer to developer, developer to Scrum master, Scrum master to product, owner and product owner to developer. The daily feedback in the first two links are well supported by Scrum. Closed loop communication with the product owner is only ensured by review meetings [Moe and Dingsøyr, 2008], but could be further supported by follow up meetings and communication.

Mutual Trust

> Definition [Salas et al., 2005]: The shared belief that team members will perform their roles and protect the interests of their teammates.

The commitment to sprints as well as protection and reliance on the team means that Scrum assumes that mutual trust is in place. Without it commitment and keeping deadlines is difficult [Moe and Dingsøyr, 2008]. There are no mechanisms in Scrum designed to develop mutual trust, which could be a challenge for new teams.

Team Leadership

> Definition [Salas et al., 2005]: Ability to direct and coordinate the activities of other team members, assess team performance,

> assign tasks, develop team knowledge, skills, and abilities, motivate team members, plan and organize, and establish a positive atmosphere.
>
> Proposition 1: The team leader will influence team effectiveness through his or her ability to set or reinforce performance expectations including performance monitoring and backup behavior.

One note on these definitions is that existing theory on team leadership is biased towards individual leadership theory, making direct comparison difficult[Salas et al., 2005]. Compared to Scrum, the leadership definition is covered by the the Scrum master and product owner roles, and to some degree the team itself.

The Scrum master, described as more as a coach, and less as a leader, is regarded by Moe and Dingsøyr as the central role in Scrum leadership. The responsibility is to make Scrum work, which compared to the definition of leadership is somewhat unspecific. The clear tasks include team protection, and daily monitoring of impediments in a sprint, but responsibilities may exceed these significantly, depending on the qualities of the team and product owner. One example is setting and reinforcing performance expectations which is done in sprint planning and sprint review. Setting these is in essence the responsibility of the team in cooperation with the Scrum master. Reinforcing these is a task primarily assigned to the product owner, but the Scrum master has a participating role.

Mutual Performance Monitoring

> Definition [Salas et al., 2005]: The ability to develop common understandings of the team environment and apply appropriate task strategies to accurately monitor teammate performance.
>
> Proposition 2: Mutual performance monitoring affects team effectiveness through effective backup behavior.
>
> Proposition 3: Effective mutual performance monitoring will only occur in teams with adequate shared mental models and a climate of trust. [Salas et al., 2005]

Moe and Dingsøyr found that Scrum does not directly support mutual performance monitoring, but enables it. The daily scrum meeting is an important practice, as well as the sprint burndown chart. Review and retrospective meetings enable discussions on what performance is expected, which is a key component as unknown performance expectations makes monitoring deviations difficult.

Back-Up Behavior

> Definition [Salas et al., 2005]: Ability to anticipate other team members' needs through accurate knowledge about their responsibilities. This includes the ability to shift workload among members to achieve balance during high periods of workload or pressure.
>
> Proposition 4: Backup behavior affects team performance directly by ensuring that all aspects of the team task are completed.
>
> Proposition 5: The effect of backup behavior on team effectiveness is mediated by the team's ability to effectively adapt to changes internal and external to the team.
>
> Proposition 6: Effective backup behavior requires the existence of adequate shared mental models and mutual performance monitoring. [Salas et al., 2005]

Back-up behavior is important to Scrum as it is incorporated into the multi-functional self organizing team [Moe and Dingsøyr, 2008]. While this aspect is important, there are no mechanisms in Scrum to ensure it functions well. Adoption of other Agile practices, such as pair programming seems to be a commonly used way to do it.

Adaptability

> Definition [Salas et al., 2005]: Ability to adjust strategies based on information gathered from the environment through the use of backup behavior and reallocation of intra-team resources. Altering a course of action or team repertoire in response to changing conditions (internal or external).
>
> Proposition 7: Adaptability of a team has a direct effect on team effectiveness.
>
> Proposition 8: Effective adaptability requires the existence of adequate shared mental models and effective engagement in mutual performance monitoring and backup behavior.[Salas et al., 2005]

Moe and Dingsøyr writes that Scrum is designed to be adaptable, by using short feedback loops and frequent re-planning. This applies both to task-related aspects as well as project related ones. Self-organization should be considered as an enabler of this, because it does not guarantee adaptability,

but ensures that the team is not bound by too much commitment to plans or contracts.

Team Orientation

> Definition [Salas et al., 2005]: Propensity to take others behavior into account during group interaction and the belief in the importance of team goal's over individual members' goals.
>
> Proposition 9: Team orientation affects team effectiveness through team members' willingness to engage in mutual performance monitoring
>
> Proposition 10: Team orientation affects team performance through team members' acceptance of feedback and/or assistance through backup behavior.[Salas et al., 2005]

Scrum team members are empowered to commit as to an amount of work as they see fit, and there is a heavy emphasis on consensus [Moe and Dingsøyr, 2008] when doing planning. Goals are supposed to be clearly specified early in the project, as well as before each sprint. The team is trusted to divide tasks internally and to commit to sprints, which can be considered to require a strong team orientation. Team protection and empowerment should on the other hand facilitate it, making this a potentially important aspect in Scrum.

Chapter 3

The Seating Arrangements

The design of a research study is a product of the initial motivation, as well as the context of the study. In this study, the wish was to study and gain a deeper understanding of how Agile methods in general, and Scrum in particular, are implemented and used in a real world environment. This presented several opportunities to the type of study possible.

The result was an interpretative case-study using ethnographic methods. This chapter will give a brief theoretical grounding of relevant research methodologies. Following this the methods used and data gathered in the study is laid out. The rationale for method choices is also discussed, as well as the approach to analysis.

Finally, the validity of the study is evaluated based on an existing set of principles [Klein, Heinz K. and Myers, Michael D., 1999].

3.1 Methodology

The methodology used in research is critical to the nature of understanding gained. There is a wide range of methodologies appropriate in information systems (IS) research.

Research in IS has historically been biased towards traditional, empirical and quantitative research [Galliers and Land, 1987]. Galliers and Land criticizes this bias which consider IS as purely within the province of technology, and argues that it is more appropriate to extend a study also into behavioral and organizational considerations. In later years, it seems that this historic bias has changed. Two of the methodologies classified in Galliers and Land [1987] as new, are the descriptive/interpretive methods and action research. According to Walsham [June 2006], these mainly qualitative methods are

now quite common and a well-established part of the field. As a side-note, research on Agile methods is biased towards case-studies. In Dybå and Dingsøyr [2008] 72% of studied found were single or multi-case studies.

Generalizing on single-case studies is a debated subject. Walsham [1995] suggests four types of generalization from interpretive case studies: the development of concepts, the generation of theory, the drawing of specific implications, and the contribution of rich insight. In addition to these four types, Rolland and Herstad argues that a "critical case" can give another benefit to generalization, namely falsification:

> Case studies are useful for falsifying existing theories, because case studies pays particular attention to context and situations that might explain why outcome of an action is inconsistent with theory.[Rolland and Herstad]

This quote can be used to illustrate the intention of this study. Scrum claims reintroduce flexibility, adaptability and productivity into systems development [Schwaber and Beedle, 2001]. Subsequently, performing a case study can be useful to explore the validity of these claims and shed light on important contextual factors.

3.1.1 Interpretative Research

Qualitative studies can be done based on a number of different underlying philosophical assumptions. The interpretative is one of these, which is based on the assumption that our knowledge of reality is a social construction of human actors [Klein, Heinz K. and Myers, Michael D., 1999]. Various artifacts such as language, consciousness, shared meanings, documents and tools are important to construct this knowledge.

Interpretative research have traditionally been an anthropological research tradition [Walsham, 1995]. When used in IS they are aimed at understanding of the context of the system, and the process of how a system influences and is influenced the context [Klein, Heinz K. and Myers, Michael D., 1999].

Social dimensions are an intrinsic part of software development, which is regarded by Sharp and Robinson as mainly a social activity. Shared values, assumptions, beliefs, and the influence of particular individuals are important to provide insight into the process. Interpretive research has been used to understand how software development teams operate, and to facilitate deeper reflection on process [Sharp and Robinson, 2004]

Ethnographic Methods

One methodology suitable for interpretive research is ethnography. This type of research is originally from social and cultural anthropology. It can be described as a method where the researcher is immersed in the study for a significant amount of time, from months to years. The objective is to place the phenomena studied in their social and cultural context [Myers]. There are several examples where ethnographic methods have been employed in information systems research, such as Sharp and Robinson [2004] and Myers and Young [1997]. According to Myers, ethnography has become widely used in research on information systems, including research on the development process.

The primary data source is detailed, observational evidence [Myers], supported by various types of material; field notes, audio and video recordings, photographs, sketches of physical layout, copies of documents and artifacts and records of interviews [Robinson et al., 2007]. When doing ethnographic research Myers suggests three general rules of conduct; that field notes should be written up on a regular basis, write up interviews as soon as possible, and regularly review and develop ideas as research progresses.

3.2 The Study

The initial with to study Agile methods in practice presented several opportunities. The single case studied, and the focus on Scrum was chosen because of two factors.

- The number of scientific studies on Scrum is limited [Dybå and Dingsøyr, 2008].
- A researcher from SINTEF was already involved in a project which used Scrum.

Working on one case with another researcher provided potential benefits for both, where sharing of research material and someone to discuss the case with were the two most prominent.

3.2.1 A Choice: Research Approach

Choosing this was inspired by Sharp and Robinson [2004], which highlighted the usefulness of using ethnographic methods when studying a development process.

Another very important factor was that the co-researchers already had adopted an ethnographic approach. Consequently , the study was decided to be interpretative using ethnographic methods.

Ethical Research

Walsham [June 2006] addresses 3 different ethical issues that are possible in interpretive research; Confidentiality and anonymity, working with the organization, and reporting in literature. These were considered in the study.

The study site, research subjects, projects names or other identifiable aspects of the study are kept anatomized, both in writing and in pictures. Further, research material that can identify the people involved is not released and regarded as confidential.

Working with the organization was included in the larger action research project this study is part of. When it comes to identifying organizational issues, the research subjects were cooperative and willing to discuss the researchers views. The company had entered the research project with the motivation of getting insight to improve their own processes.

With regard to reporting and possibly portraying the organization negatively, this was handled by giving the people involved the possibility to comment on the findings. A presentation and feedback-session was held two weeks prior to the submission, and the company also received an unfinished version of the document less than one week before the deadline.

Bias

It is important to be up front previous experience and expectations before starting a study.

During the study, the author was a student in informatics, thus the experience with software engineering methods was mostly from education. The initial take on Agile methods was generally positive, and this was reinforced by studying theory on the subject before the study. By these two factors, the author recognized an initial slight bias towards Scrum. However, this bias was negated by limited practical experience.

3.2.2 Fieldwork: Style Of Involvement

Access to the study site was negotiated as part of a larger action-research study project, EVISOFT which is a national industrial project on process improvement. When the author got involved in the case (middle of March)

the co-researcher had been on-site since mid February and was working with the team on design tasks.

The roles in this study varied in a spectrum of roles from the "involved" to the "outside" researcher [Walsham, June 2006]. In the beginning of the project the involved researcher was the dominating role. The co-researcher worked extensively with the team on GUI-design. Later when the author became involved, both were active in workshops on high-level design, identification of concerns and estimation and planning meetings.

In the two first development sprints, the author had responsibilities in running unit tests, and also did some work on setting up automated testing and building. The Scrum master often asked in daily scrum meetings how testing was going. In the later parts, this participating role disappeared altogether, and was replaced by a more passive role.

The co-researcher was in the development sprints generally used as a consultant on process and did not have any directly development related responsibilities. Among other things he was responsible for introducing the index card wall in the project. In the following quote the Scrum master addresses the co-researcher in a way which illustrates his position:

> Scrum master: Remember to let me know if you come up with something clever!

From sprint 3 and onwards, both researchers became more outside researchers which attended but did not participate in meetings. As figure 3.1 shows, the number of contacts decreased towards the later stages. Also the median duration of contacts decreased.

To summarize, our style of involvement was inductive. It is perhaps fitting to say it evolved with our understanding of the case. In the early stages much time was invested in working with the team. As the team became more involved in the work and interesting concepts became clearer, the information gathering process became more focused towards the research questions.

3.2.3 Data. Lots Of Data

The data collected was from three distinct sources; material from observations, documents and charts from the computerized process support, and interviews.

Field notes from participant observation, pictures, documents and other types of material acquired using ethnographically informed methods formed the majority of material gathered. The gathering of the material is summarized in figure 3.1 which shows the distribution of contacts along the weeks

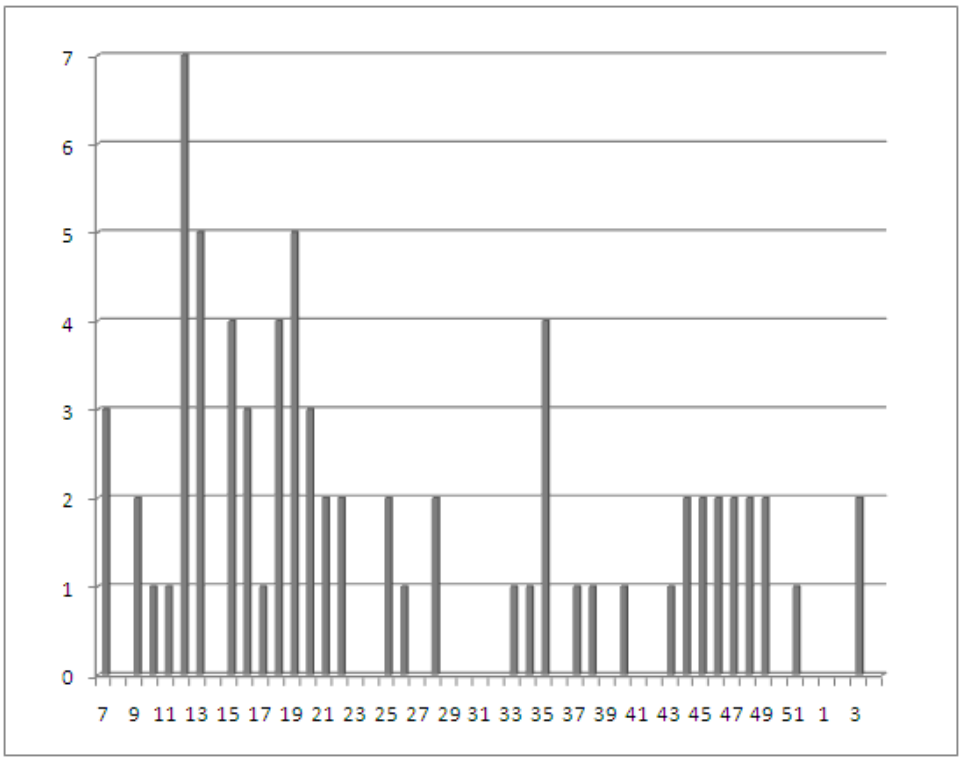


Figure 3.1: Graph showing the distribution of contacts per week.

of the study. A contact in this context is defined as documented communication between researcher and subject. There were additional meetings, such as a conference in June, but these were not documented, and is not included in the summary. Table 3.1 shows the number of distinct contacts ordered by type throughout the study.

As stated earlier, field notes was the major part of material gathered. The size and scope of these field notes varied, from the short impressions after a quick visit, through detailed summaries of meetings, to transcriptions of discussions. In some of the field notes illustrative quotes was recorded. The quality also varied much. Some of the earlier notes were, to be honest, very light on information because of inexperience and lack of focus. However, the quality improved significantly in the later notes. Pictures was a valuable addition, and a digital camera was used when appropriate. Other documents and artifacts included some of the index cards used in sprints.

A lot of supporting material was gathered from the project support system used (see section 4.3.3 for more details on the software). Among the types of documents were burndown charts, sprint and product backlog composition, sprint views (sprint backlog with the estimated remaining effort for each item

Type of contact	#
Participant observation	23
Daily Scrum meeting	19
Sprint review meeting	6
Sprint planning meeting	4
Retrospective meeting	1
General meetings	7
Other (email, telephone etc.)	6

Table 3.1: Distinct contacts through the study.

through a sprint) and other Scrum-related reports. Various types of project documentation was also gathered from this system, as these documents usually were stored in the integrated content management system. In addition, this software suite featured data-warehouse capabilities on source-control, and this was also used to some extent. Some of these data are included in appendix A.

The final data source was interviews of the 3 developers and the Scrum master. These interviews were performed in the middle of December at the study site. Each interview was aimed to last approximately 30 minutes, and the interview guide (included in appendix B) was sized accordingly. The author and the co-researcher performed the first interview together and then did the remaining 3 individually.The interviews were then transcribed by the author and put into a summary sheet where the answers from all of the interview subjects could be easily compared.

In retrospect, some additional material could have been helpful. E-mails between developers and the product owner, personal notes taken by the Scrum master, and data from the company's time-keeping system could all contributed to a better understanding of the process. However, the amount of data gathered is quite extensive and one have to say stop at one time.

3.2.4 Use Of Theory

Walsham [1995] suggests 3 distinct examples of how theory can be used in a case study.

1. As an initial guide to design and data collection
2. As part of an iterative process of data collection and analysis
3. As a final product of the research

This study uses theory in both role 1 and 2, which is related to the two specific research questions; RQ1 and RQ2.

In RQ1 the idea was to study a Scrum project and compare the process to existing theory on the subject. This did from the authors perspective mean that it was important to have a good knowledge of theory before starting the study. Notable theory includes general studies on Agile methods such as Abrahamsson et al. [2002], Boehm [2002], Highsmith [Sep 2001], as well as more Scrum-specific material [Schwaber and Beedle, 2001, Fitzgerald et al., 2006].

Walsham [1995] warns that this use of theory presents a danger of seeing only what the theory suggests. This warning was given weight throughout the study, and it was emphasized to keep an open mind to "deviations" from theory. It should be noted that the action-research parts of the study, meant that the researchers probably influenced the implementation somewhat toward a more strict implementation of Scrum. This use of theory does however not necessarily mean that theory was treated rigidly. It was rather an initial starting point for all of the parties involved which facilitated further work, and the author firmly believes that the findings confirm this.

The secondary research question, RQ2, was adopted in the later stages of the study, and was directly linked to the framework in Salas et al. [2005] on teamwork. This framework had previously been employed by my co-researcher and my supervisor on other cases, and using it in this case could be useful for comparison and presented opportunities of multi-case publications. The analysis done when deciding to use this theory, revealed that our data was inconclusive on some of the aspects of teamwork discussed. With this in mind, the interview guide used in the interview had a few questions angled towards how teamwork had functioned in the project.

3.2.5 Analysis

Analyzing interpretative research is a process which often requires some time and is described as a cycle of data gathering and analysis Walsham [June 2006].

Walsham [June 2006] writes that analysis of interpretative data when using theory to guide the research, often leads to the researcher to understand and form data-theory links while doing the study. Compared with this study the description fits well as theory on Scrum was the initial starting point and guided further data gathering as understanding evolved. Continuous analysis was done through frequent discussion with the researchers involved in the case, namely the author, the co-researcher and the supervisor. Discussions which were valuable to guide and improve data gathering. The amount and variation of data gathered did however require a more systematic analytical approach in the later stages in order to better understand the case.

In October the field notes then gathered was coded thematically using a software suite designed for qualitative research [1]. While this was a valuable way to read through the data, the themes identified was not used in the later stages of the analysis and was not refined further. The reason was that this way of handling the data felt too rigid, much like the lock-in effect described in Walsham [June 2006]. Coding all of the data thematically was a very time-consuming process which was difficult to justify by the insights gained.

A different approach was then tried. Inspired by the contact summary sheets in Miles, Matthew B. and Huberman, Michael A. [1994, pages 54-56] the material was processed and distilled into weekly contact summaries. The approach described differs somewhat from the actual summary forms used; One difference is that the summaries are supposed to be based on write ups of the raw data, while they in this case were based on the raw data alone. The described approach also states that the summaries should be at the most one page, while some of the finished summaries were 3 or four pages.

These differences probably contributed to making the summaries more complex than the example summary forms provided, but they were useful in the same way as described; as practical way to do first-run data reduction without loosing too much precision. The summaries used the following template:

- Encounters - A list of the weeks contacts; date, researcher and nature.
- Summary - Summary of the data gathered that week.
- Events - Notable events.
- Process - An evaluation of the process used according to theory.
- Key observations - The most important observations.
- Quotes - Notable quotes from the material.
- Pictures - Notable pictures taken.

Compared to thematically coding the raw material, this technique was perceived by the author as much more open-ended and valuable. While some precision is always lost using such a technique, the source material was used as a reference when needed. The summaries was used as a source when doing the final analysis.

Further analysis was then done while iterating between writing this thesis, reviewing the summaries and more writing.. Discussions with the other researchers were also common, and an analysis-workshop was held where the case was analyzed thematically using post-it notes and whiteboard (see figure 3.2). About three weeks before deadline, the author had a presentation and feedback session for the team and other interested parties in the

[1] NVivo7 by QSR software

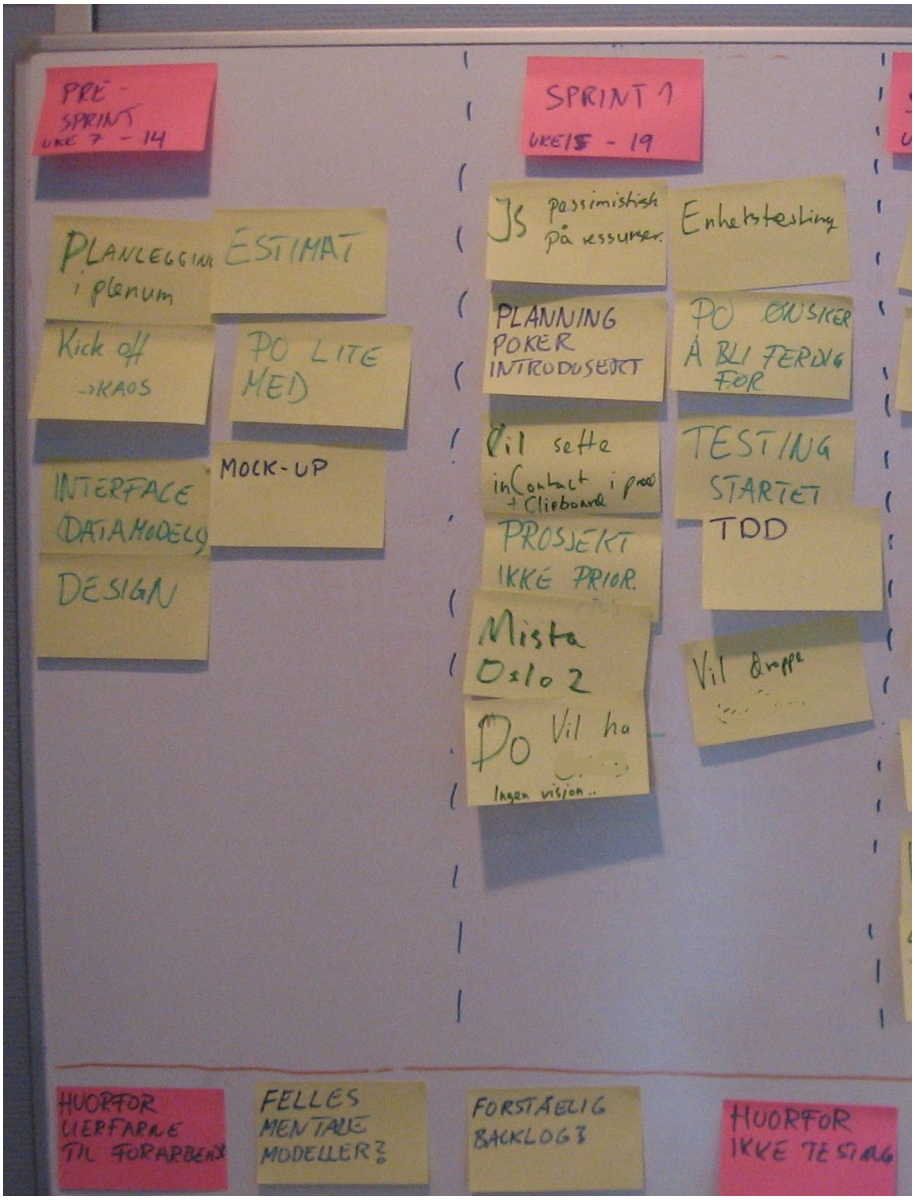


Figure 3.2: Analyzing with post-its and whiteboard.

company. The subject were the preliminary findings, and some issues were discussed.

While the analysis probably could have been done more systematic, the following quote from Walsham inspired and guided the process, and is perhaps a fitting summary of the analysis work:

> I believe that the researchers best tool for analysis is his or her own mind, supplemented by the minds of other when work and ideas are exposed to them [Walsham, June 2006].

Chapter 4

Match Day: Live Scores

This is the main chapter of this thesis, where the actual findings are presented. The aim is to give the reader an understanding of the study, emphasizing the details relevant to the research questions.

It is divided into three sections; the first section presents the project and surroundings, the second section give a chronological overview of the project, the third section describe the development process and the implementation and use of Scrum.

4.1 Background

The study took place in the development department of a medium sized Norwegian company, studying the development of a system for use internally.

As the organizational chart in figure 4.1 shows, the company had 3 regional divisions with one separate ICT division. These regional divisions, with offices scattered throughout most of Norway, was the actual customer of the project, and represented the company's core business. The ICT division, consisted of a consulting department, an IT management department, and a development department. These departments main business were development and maintenance of a series of off-the-shelf software products developed in-house. They were also reseller of some products. Additionally, the development department did projects for external customers.

4.1.1 The Development Department

The development department had approximately 16 employees, divided into a Java and a .Net department. Generally, the atmosphere in the development

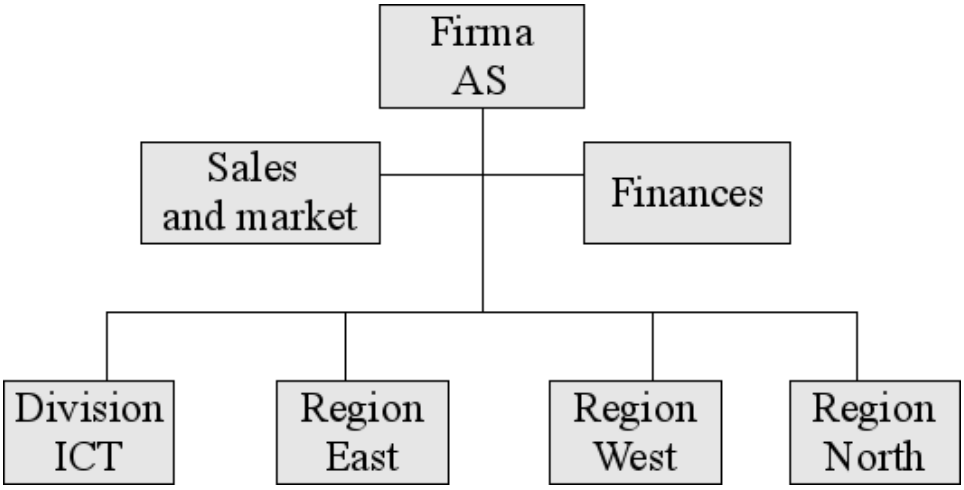


Figure 4.1: Organizational chart

department was informal and relaxed with little visible internal hierarchy. The department head (and also project Scrum master), was located in the same open-plan offices as developers.

The open-plan offices were a relatively recent introduction. The department also had some experience with Scrum as it had been used on an earlier project, with mixed results. During the study there were started two new projects using Scrum, and there was generally much interest in using Agile methods among the developers.

Scrum is advertised to be effective in handling noisy projects [Schwaber and Beedle, 2001], and categorize noise along two axes, technological and requirements. In this project, these two factors were relatively stable. There was however much noise in the project, but this was related mostly to the surroundings of the project. Especially two practices were problematic, and these are described in the following sections.

The Quagmire

The quagmire is a term which was on a few occasions used by the developers to describe some aspects of the company culture. These aspects were in large part related to the lack of a dedicated software support and management department, but also to the number of projects. The quagmire describes when a developer is stuck with supporting functionality developed in a earlier project, and is forced to use time on this even when engaged on other projects. One of the developers explained it in the following way:

> Developer: Over time it has become common, that if you take re-

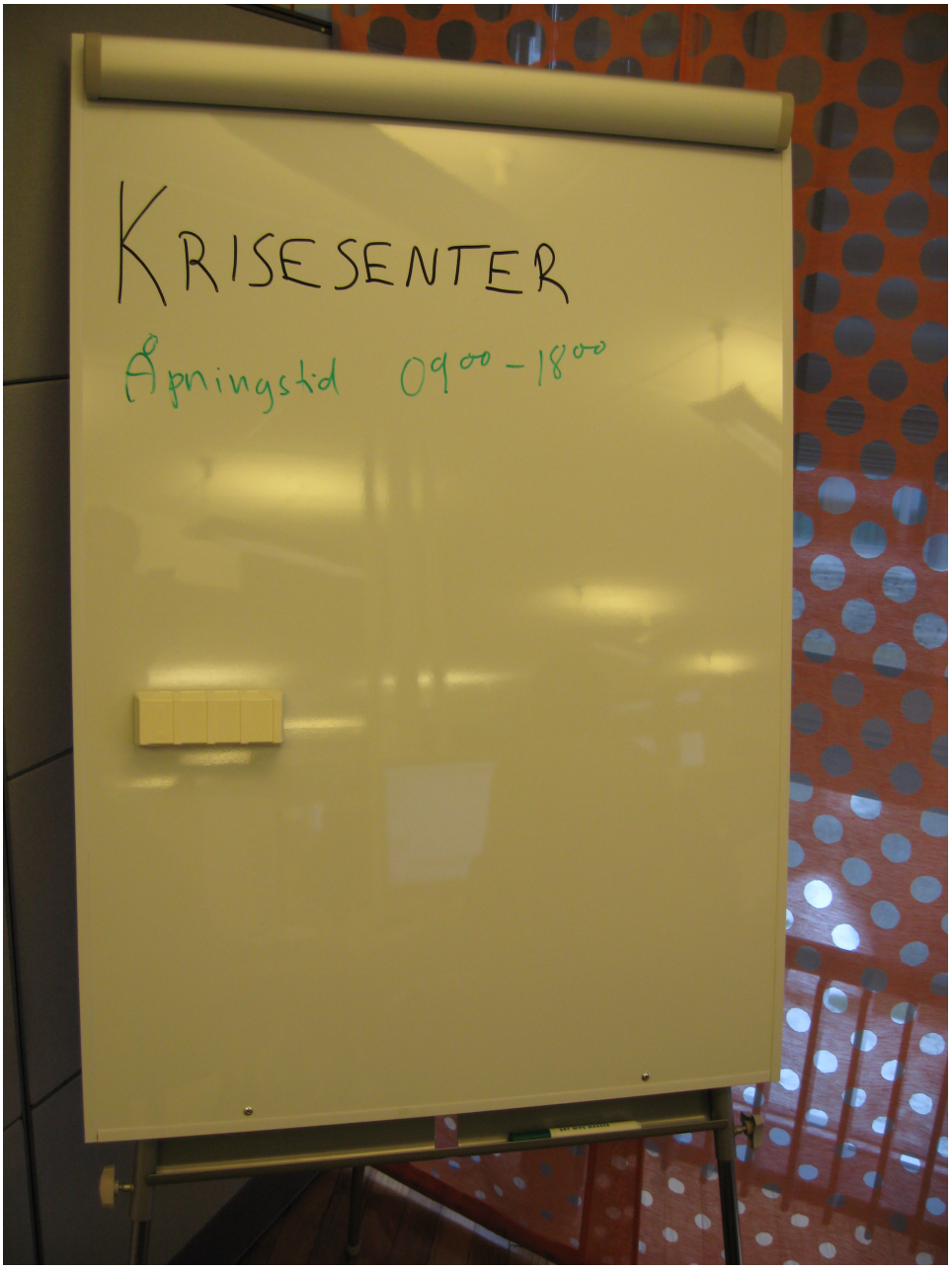


Figure 4.2: A humorous take on the quagmire. The Whiteboard reads "Emergency center"

> sponsibility for something, then you are stuck with it for eternity. This can lead us to avoid responsibility.

The developers felt this situation was difficult. They would often use time responding to requests from earlier customers, because they were the only ones capable of doing so. Another developer coined it the following way:

> Developer: I don't think management knows how much time we spend on the changes we perform. And I don't know if the company is paid for work done on maintenance and changes. As an example, related to the SMP-project. It has been a continuous emergency situation the last period. It is a lot of work, continuous fire fighting.

Also the Scrum master, who was head of the development department was frustrated with this situation.

> Scrum master: Generally, it is customer follow up; support work that consumes time, especially so with Greg and Allison. There are some customers that are completely dependant on support. We don't have any other department that can handle this.

There was talk about establishing a separate support and management department, but this did not materialize during the project.

Parallel Projects

It was quite common that developers were involved in several projects at the same time. Only newly hired developers were spared of this.

This situation affected the developers in some ways. One was that frequent meetings, customer contact and noise from other projects meant that they were occupied with much more than development. There probably existed significant overhead, which sometimes would leave little time for coding.

> Developer: I get really frustrated when I barely can write a single line of code.

Projects that mostly were finished was also a real concern for the developers previously assigned. One of the developers noted the following about a project which was being implemented at the customer:

> Developer: I feel that this project is going to be long, it is gradually turning into a nightmare.

This project was essentially finished, and the rest of the developers had ceased to work on it. However, there still remained work, and getting assigned to a new project still meant that this had to be done. Through this,

the developers felt that it was difficult to get the opportunity to finish, proceed and focus on new projects. Previous project managers and customers would still interfere and give assignments.

The product owner was also aware of this situation. In one of the review meetings, the product owner jokingly expressed that he wished the project was on a fixed-price contract. This illustrates that while the project was prioritized by management, it still competed with other projects on resources. As an internal project, it was sometimes more important to priorities outside customers, a view which was reinforced by the developers.

4.1.2 The Project

The study object was a project doing the redevelopment of a software package used internally in one of the company's core areas, The operators of the software used it in day to day operations when dealing with the customers handling reports used in coordination activities. The volume of reports are linked seasonal constraints. Winter is the low season and the high season begins in March/April. The seasonal constraints gave a relatively narrow time frame for introduction of a new version of the software, and a quite firm deadline for completion of the project which initially was January 1, 2008.

The old version (version 3) was developed and maintained by a consulting firm. Version 3 was relatively mature, but the company wished to maintain and develop the software internally. The motivations was to take control of core systems, and to improve the software used by operators in order to reduce the time used on handling a report. Management decided to develop a new version from scratch in-house using newer technology[1], and the project can subsequently be called a re-enginering effort, although with new features added.

Curiously, the project never had a proper vision statement for the project, even when the Scrum master asked the product owners several times to create one. "We want to create a system that can grow" was at one time mentioned, but this was never used as the official one. Unofficially, the intention of the project was to create a system which had the same functionality as the old one, but did the same tasks more efficiently and added certain functions.

As previously described, the development department had several parallel projects. According to the team the project had a high priority from management, but was prioritized higher towards the later stages. This picture

[1] The new software is developed on the .NET platform using Microsoft Visual Studio Team System, from now on abbreviated VSTS

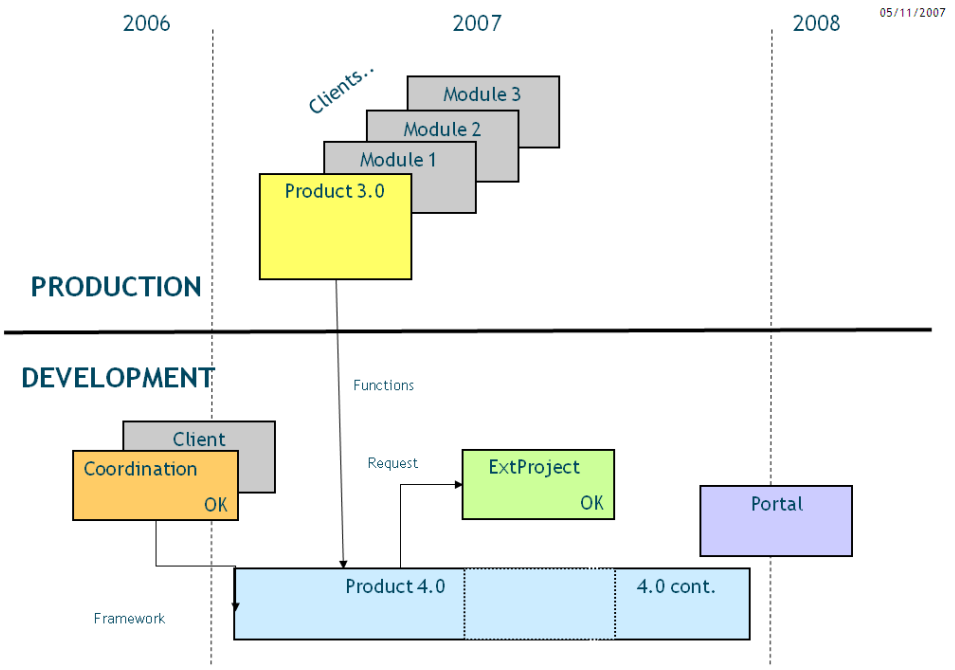


Figure 4.3: Part of presentation showing some of the different projects in the company

is consistent with other data; In the early stages, comments from the product owners as well as the priorities in the first sprints, suggests that the early stages was in some respects a proving ground to see if the project was worth going for. Figure 4.3 reinforces this view. The changes done in the later stages further suggests increased priority, and supports the statements about priority increasing towards the end.

Prioritizing a project over another was difficult, and the Scrum masters dual role probably also interfered. At one time, another project was put into production where two of the developers from the project were involved, and consequently the progress on slowed down leaving the 100% developer more or less alone. There were also several times where other projects were discussed in the daily Scrum meetings.

The Team

Initially the project consisted of 2 developers and 1 scrum master (see table 4.1). Also, two product owners were involved[2]. The project had a man-

[2]When discussing the product owner role, the plural form is used in situations where both product were involved, as in review meetings, and single form when referring to

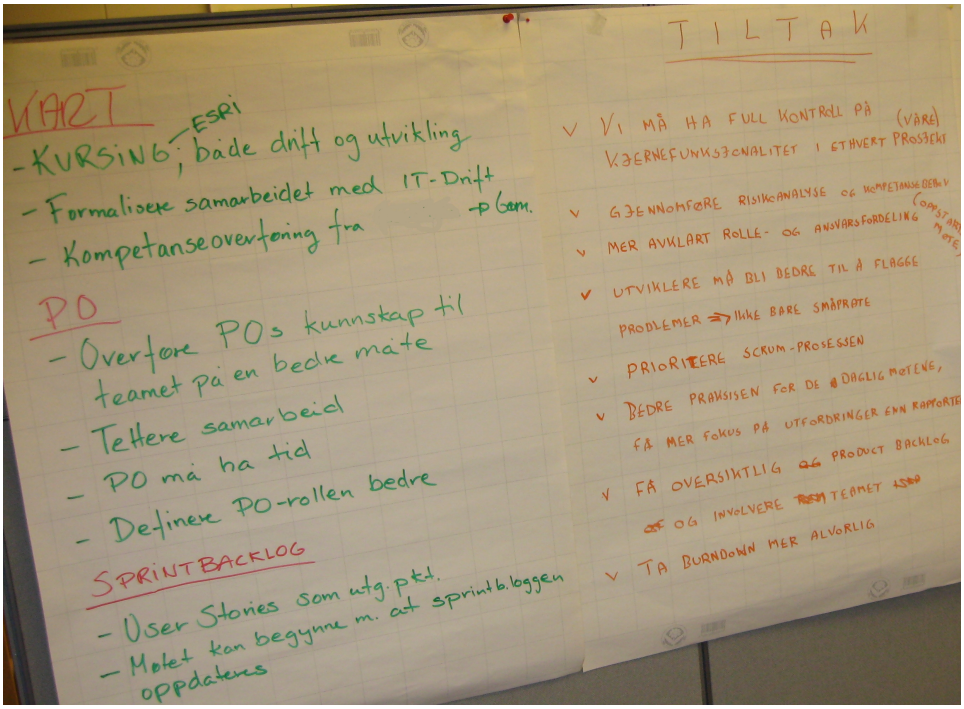


Figure 4.4: Experiences from a retrospective after the previous Scrum project.

agement group, overseeing progress and budgeting on a higher administrative level. A third developer joined the team late in sprint 1 [3]

Scrum Experience

The teams previous experience with Scrum influenced the project. Two of the three developers, as well as the product owner participated in an earlier Scrum project. The experiences from a retrospective after this project is shown in figure 4.4, which lists a number of concrete measures to improve the process. This project had been a turbulent one with regard to Scrum, and the developers referred to this on many occasions.

The Scrum master did not have previous experience as a Scrum master, but had access to the experience reports and results from retrospectives in the previous Scrum project. With regard to training, the entire development department had a 1-day Scrum-workshop led by SINTEF prior to the project,

Product owner 1 (see table 4.1).

[3] Team is used about the developers and Scrum master.

Name	Description
Scrum Master	Scrum master for the project as well as manager of the development department. Has a technical background, and participates in some development work towards the end of the project. Attended Scrum master certification with Mike Cohn in September.
Developer 1	The main developer on the project, and the only one with close to 100% time allocated. Was newly hired in the company when the project started, and graduated with a masters degree in 2006. Worked on a start-up after graduation.
Developer 2	Have 5 years of experience in the company. Was generally involved about 30-40% on the project, more in the last 3-4 months. Participated in an earlier project which also used Scrum, and was working on two parallel projects in addition to this. Became a certified Scrum-master halfway in the project at the same course as the Scrum master.
Developer 3	Was the last developer added to the project, and started working on it in sprint 1. Used about 30% of time available on the project. Has worked in the company since 2005, and worked on with Developer 2 on the earlier Scrum project, which he also worked on during the study. Was also involved in the same two parallel projects as Developer 2.
Product Owner1	The main product owner, who the team communicated the with. As he is located in another city, teleconferences were used when he was unable to attend important meetings such as sprint review.
Product Owner2	Was involved in some of the earlier work, and has been to most of the review meetings. Is based in another city. Non-technical background, but has experience with the old version of the software, and has been involved with several earlier development projects. Is in practice more a project manager than actual product owner.
Development Chief	Certified as Scrum master along with the Scrum master and Developer 2. Was generally not an active part of the project.
Regional Manager	Part of the management group for the project. Attended at least one review meeting.

Table 4.1: Key actors in the project

which he attended. In sprint 4 the Scrum master together with one of the developers finished a Scrum master course.

Engineering Practices

Scrum does not specify engineering practices, and can be used to encapsulate the existing practices. Encapsulation was also the main theme in this project, although some new practices were adopted as it progressed.

Automated building of the system was employed throughout most of the project, except from the first two sprints. In sprint 3 the team started automated building of the system, through the integrated build system in VSTS. A build was started when a developer checked in code, and the development team was then notified by email of the status of the build. The main reason for this was said to be a need for improved synchronization when the team worked on overlapping parts of the system.

In most parts of the project, there was little organized user testing. Unit testing was tried to some extent in sprints 1 and 2, but was not carried out at later stages. The developers would test while developing, demo, and fix what was commented in the in review meetings. The product owner did also do some testing during the sprints, and gave feedback through email, telephone or other means.

The developers indicated that the approach described was the usual way to do testing. Although the Scrum master wished to introduce unit testing, it was not really adopted by the team. The reasons are unclear. One of developers expressed that he found it hard to create proper unit test, suggesting that lack of training was a factor. However, it was probably a lack of interest and perceived benefit that prevented unit testing from gaining traction. As one of the developers put it:

> Developer: We have managed without testing for many years, so it should be OK.

The Setting

Scrum advocates open-plan offices to facilitate communication between team members [Schwaber and Beedle, 2001]. This was also the case in this study. The entire development department, of approximately 16 persons, is located in an open-plan office with 4 separate islands with 4 desks per island. The team members on this project were co-located on one of these islands. The Scrum master sat in the 5 first sprints at another island, but moved to sit with the developers from sprint 6.

Figure 4.5: The teams "island" in the development area

In addition to the actual desks needed for day to day development, a meeting room was used to conduct the daily Scrum meetings, review meetings and other collaborative activities. This meeting room was down the hall from the development, and was shared with the rest of the company. It had a projector, equipment for teleconferencing, and seating for 8-10 people. In the later stages of the project, a Nintendo Wii gaming console was hooked up to a big screen TV and set up in the development area. The developers, and sometimes people from other parts of the company, occasionally played Wii:Sports game on this during the working hours. At one recorded occasion, the division head joined the developers for a game of tennis.

4.2 Project Overview

The project started in February 2007 and was scheduled to finish development by end of the year. As it progressed the use of Scrum evolved considerably, and viewed in hindsight, was done in four distinct phases; pre-game Scrum, everyday Scrum, emergency Scrum and post-game Scrum.

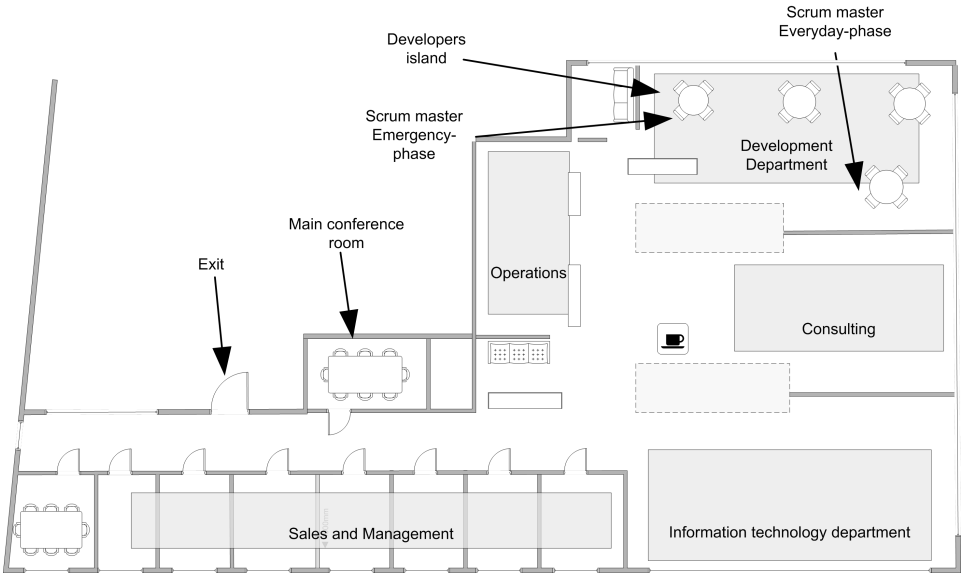


Figure 4.6: Office floor plan

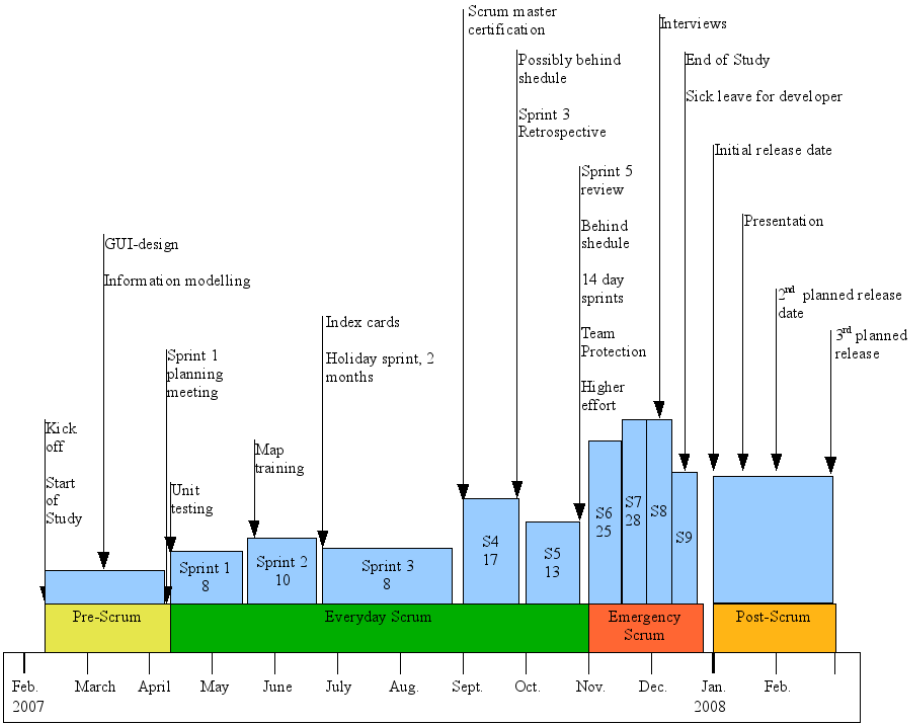


Figure 4.7: Timeline of the project with some important events. Effort is visualized by the height of each sprint, with average hours per work day under the sprint number.

Ergonomic factors	
Physical Layout	Open space, team in same desk island
Distracton level of office space	Medium
Customer communication	e-mail, tele conference, meetings
Geographic factors	
Team location	Collocated
Customer cardinality and location	Two, one primary, both in another city.
Supplier Cardinality and Location	Multiple; both remote and local
Sociological factors	
Team size (developers)	1+2
Team education level	Masters: 3
Experience level of team	<5 years: 3
Domain expertice	Moderate/Low
Language expertice	Low/High
Experience of Project Manager	Moderate
Specialist available	None
Personell Turnover	33%
Morale factors	Many parallel projects
Project specific factors and Software classification	
Software classification	Information system
Delivered user stories	214
Domain	Industry specific
Person Months	25
Elapsed Months	10
Nature of project	Reengineering with enhancement
Relative Feature Complexity	Moderate
Project Age	0
Constraints	Date Constrained, Resource constrained

Table 4.2: Context factors in the study. Influenced by Williams, L. et al. [2004].

To follow up on the football metaphor, the team planned their tactics prior to the match and played a semi-decent first half. At half-time they realized that they were behind 1-0, and decided to step up their efforts. In the second half of the match, they equalized and finished 1-1. As the study ended, the match went into overtime.

4.2.1 Pre-Game: February To April

At the start of this study, in February, the project had just started the pre-game phase, and went through early analysis, design and planning. At this early stage, the future of the project depended on a bid the company had on another project. As figure 4.3 shows, the project would halt or downgrad the effort for a few months if the bid was won, and then restart in the autumn.

The vision for the pre-game phase was as follows:

> Describe the user interfaces which will need change

This vision highlighted that the project in essence was a re-engineering effort. Most of the user interface and functionality was based on the previous version, but some parts of this was to be improved.

To do this the team employed white-board and paper based prototyping combined with a digital camera to create and document the interfaces (see figure 4.8). In addition, some diagrams were made to better understand the workflow.

This work was done by one of the developers and the Scrum master in co-operation with one of the researchers. The main source for this work was a document created by the primary product owner, describing the interfaces that needed change. Questions was generally directed to one of the product owner by mail. Apart from this, there was limited user interaction.

After finishing most of the interface design, the project started a 14 day long 'design sprint', where one additional developer and one researcher became involved. The tasks performed in this were: high level architectural design, data and information modelling and identification of risks and concerns. A number of concerns were identified. Among others were performance, external dependencies and development resources. To quote one of the product owners when reviewing these:

> Product Owner: I'm always pessimistic with regard to getting resources. This I know from experience.

During the design-sprint, the product owner created the initial product backlog. The basis for this was the user interface sketches created earlier, and

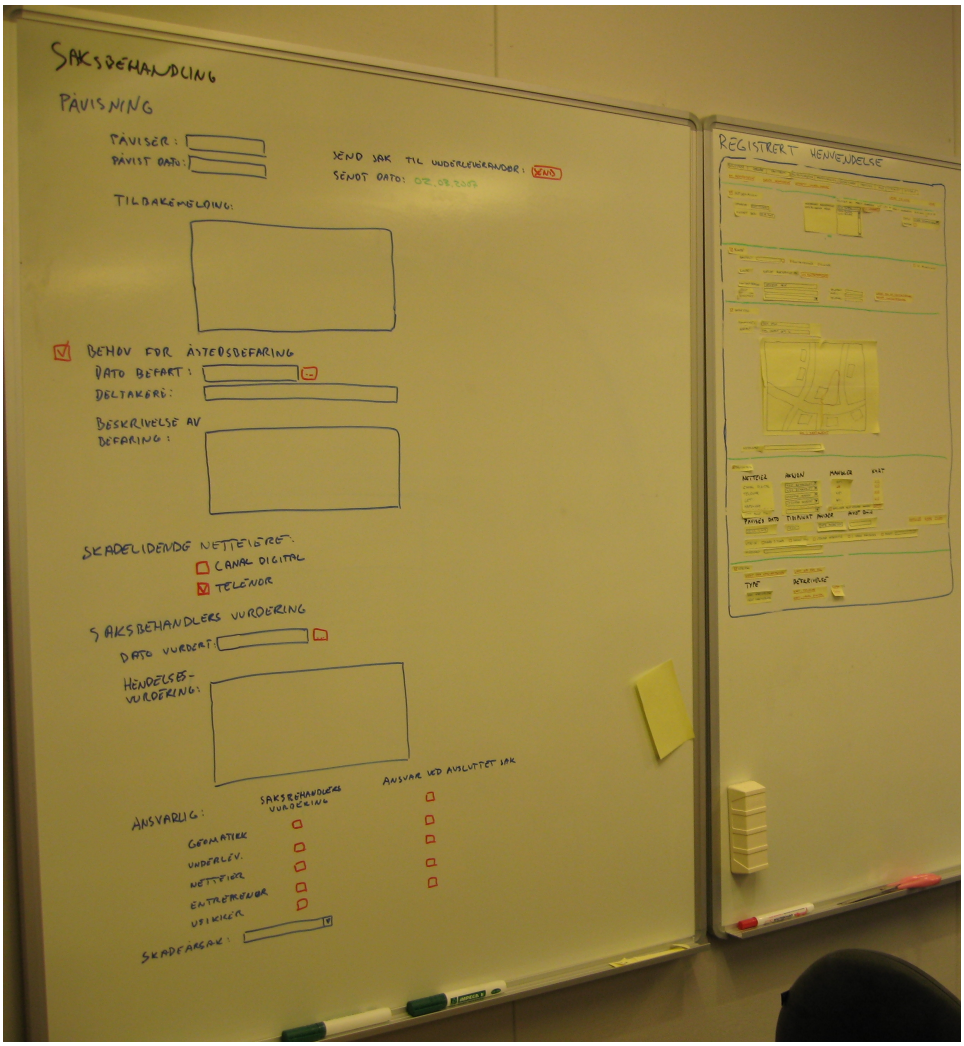


Figure 4.8: Interface design using whiteboards

this was reflected by the finished backlog. The initial backlog was quite long, and had large variations on the granularity items.

As mentioned earlier, it was at this point not certain whether the project would be started or not. This was due to the competing project but also because the scope of the project was unknown. The management group wanted the Scrum master to give an estimate on the total effort required. To create this, all items in the product backlog was estimated using planning poker, creating a total of about 3000 hours. These initial estimates were the basis for the planning of first sprints, although they were re-estimated later in the project.

At the end of the design sprint, the team received the initial product backlog which was then estimated (see section 4.3.1. Also, a preliminary release plan was created, and together with the product owners, the team performed a planning meeting for sprint 1.

4.2.2 Everyday Work: April To October

From the start of sprint 1 in April, to the end of sprint 5 in October, the team worked on the project using 30 day sprints. Progress was generally OK, and both the Scrum master and product owners were reasonably satisfied at the review meetings as pieces of functionality was delivered.

In sprints 1 and 2 the team used some time to explore the capabilities of the new project support software suite. This suite integrated most aspects of the projects into the developers IDE[4]. This exploration included engineering aspects such as unit testing and automated building as well as process tracking (see 4.3.3).

One recurring theme in this phase, was that the team never actually finished any of the sprint backlogs. This was usually planned beforehand (see section 4.3.2 for a more in depth discussion of this). The burndown charts in figure 4.9 and 4.10 illustrates this tendency. Interestingly, sprint 4 was regarded by the team as one of the best sprints, but as figure 4.10 shows, the sprint was far from finished.

The priority in the first sprint was on creating functionality which could be integrated with the existing version of the system. The team was not thrilled with patching the old system, as they wanted to get on with the new one, but accepted the product owners priorities and produced modules which was then tested and put into the production environment. The name of sprint 1 is a good illustration of this:

[4] Integrated Development Environment

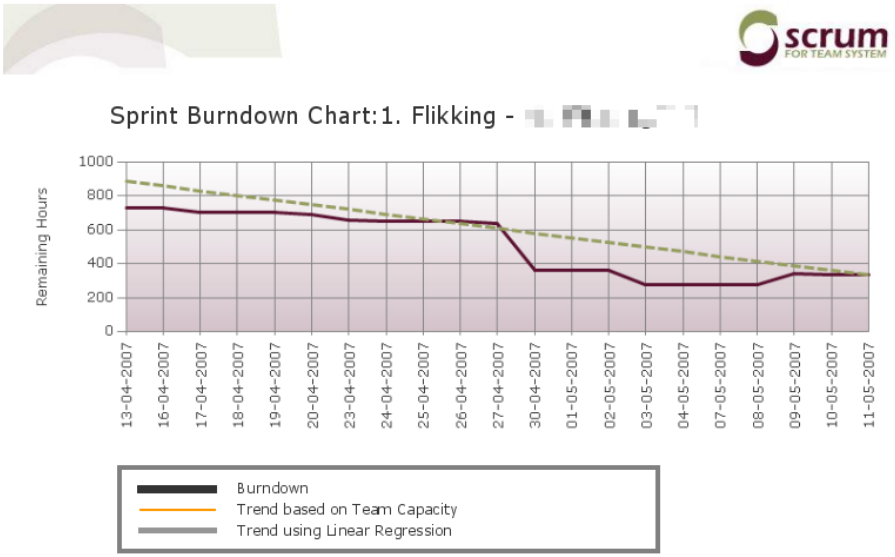


Figure 4.9: Sprint 1 Burndown Chart. Unlike theory, this sprint never reached 0 hours left.

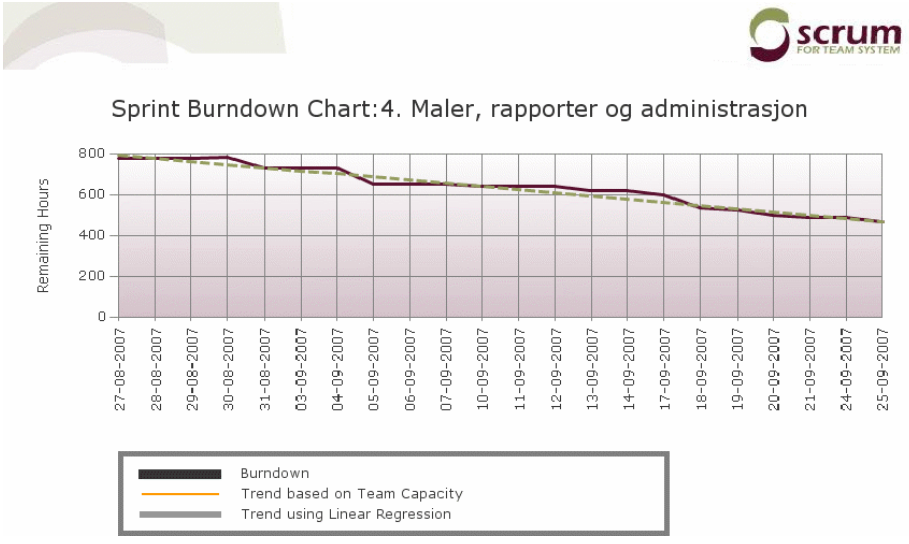


Figure 4.10: Sprint 4 Burndown Chart. While regarded as one of the better sprints, also this failed to reached 0 hours.

> Sprint 1: Patching - GSys3

Beyond this initial priority, the product owners had relatively few concrete priorities on functionality. It is perhaps best summarized by the following quote:

> Scrum master: Do you have any sensible views on priorities beyond what we have already discussed?
>
> Product owner 2: No, beyond this we want you to work in a sensible sequence.

As described in 4.1.1 two of the developers were involved in other projects. This affected priorities, especially in this phase where there generally were little protection of the developers from other tasks. The other projects also had emergencies, major releases or other things which meant that these two developers had to down-priorities the project.

This situation was planned, but often meant that the project had even fewer resources than planned initially. Sprint 1, 2 and 5 was impacted by this, as these sprints used only 60 - 75% of the originally allocated hours (this is according to the number of hours reported by the Scrum master, see appendix A). On average, the project used only 82,5% of the resources planned in the everyday period. This also made the full time developer responsible for the majority of efforts on the project.

Towards the end of sprint 4, it became apparent that the everyday pace might be too slow to finish the required functionality on time. This was discussed when planning sprint 5, but no measures were taken. The team hoped that if everything went well in sprint 5 it would still be possible to finish as planned. Sprint 5 did however not go as planned, with significantly less progress than hoped for. The following quotes are from the review meeting after sprint 5:

> Scrum Master: If we had the resources we had when we planned the sprint, things would have been OK.
>
> Scrum Master: It has been interferences from others outside of the project that has stolen time.

4.2.3 Emergency Scrum: October To Christmas

In the sprint 5 review the team and product owners knew that they could not finish the required functionality on time with the everyday pace. They had to finish the product backlog in 2007, because of the seasonal constraints. Based on the earlier sprints this was unrealistic. To quote ScrumMaster:

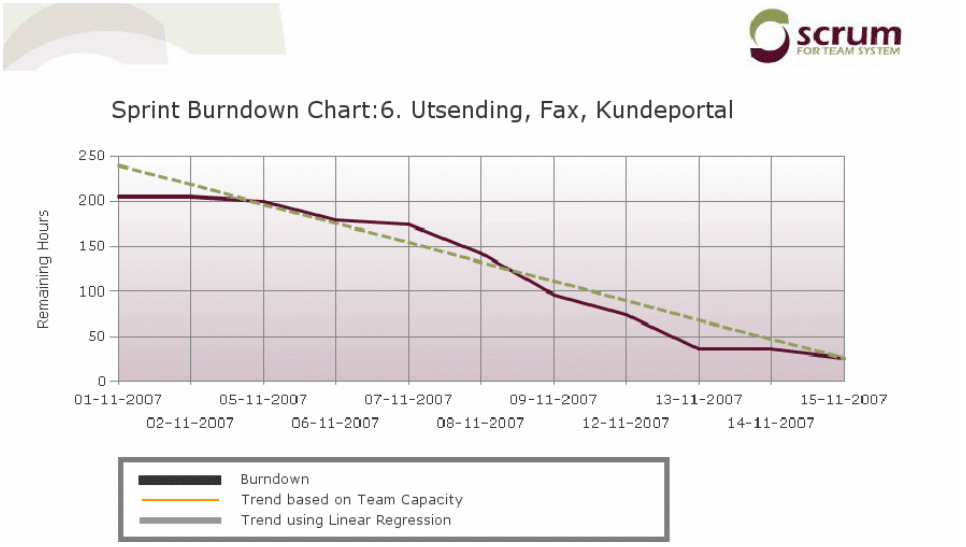


Figure 4.11: Sprint 6 Burndown Chart

> Scrum Master: We need to prepare for the fact that we probably won't finish by sprint 6

The product owner was somewhat surprised that this was an issue:

> Product Owner: This was an unexpected problem. Actually, it was two unexpected problems.

What the product owner meant by two unexpected problems is uncertain. However, after considering the options available, and discussing these with management, the following measures were put into action starting sprint 6.

- 14 day sprints which gave 4 sprints before Christmas. The motivation was to increase control and throughput.
- Team protection which involved notifying other parts of the company and protecting them from interruptions 3 days a week.
- Relocating the Scrum Master to sit with the team and use him as a developer when available/necessary.
- Overtime; not forced and kept within reasonable limits.

These measures worked well, according to some of the team members they worked better than expected. As figure 4.11 shows, the 14 day sprint 6 almost completely finished the sprint backlog, which was comparable in scope to what was finished in the earlier 1-month sprints.

The developers reported an increased sense of urgency in the project at this time. The most important change was the increased use of time, which

the team members could spend mostly on the project, and work relatively uninterrupted. Overtime was also used to some extent, and this probably improved progress further.

The interruptions didn't stop completely; sometimes because the developers felt obligated to respond to a request from a customer, or because they were assigned tasks (sometimes from the Scrum master which also managed other projects). When compared with the everyday situation it was however a big improvement. The protection worked best when the Scrum master was in the area, but they also felt it easier to say "no, I'm busy" when asked about something.

As progress improved, the introduction date became more feasible. There was however very little slack. Towards the end of the year there was still items left in the product backlog, but the most important parts were completed. In sprint 8 one of the developers had an accident which resulted in a sick leave for the remainder of the year. This reinforced the general feeling among the team that although the system could be delivered on time, some parts would still require some work. One of the developers explained this in the following conversation.

> Researcher: Do you think you will manage to deliver what you are supposed to?
>
> Developer: I think that when we reach that date, then we have something, but I believe that there still remains a long list of things that are missing or that doesn't function very well.

This feeling was probably shared to some extent with the product owners, and there was talk in review meetings on a having later releases; 4.1, 4.2 and so on. The following conversation from the sprint 7 review meeting provides some insight in this:

> Product owner 1: Can we conclude that we will get a system before Christmas?
>
> Scrum master: You will get a system before Christmas.
>
> Product owner 2: The question is what kind system we will get.
>
> <laughter>

The system was still relatively untested, and user testing started in the middle of December, about a week before the Christmas holiday.

In the start of the emergency-phase, one of the developers expressed concern that the focus on process would decline towards the end of the project, giving way to more individual and uncoordinated work than earlier. It is not evident that concern did manifest itself, but the burndown chart for

sprint 8 (see appendix A) indicates that the team stopped writing down tasks in the computerized system. One thing that seemed to function also toward the end was cooperation in the team, based on statements made in the interviews as well as observations.

4.2.4 Post-Game: January And Beyond

The study ended before sprint 9 was finished. The project was not finished, but entered the post-scrum period shortly after. The plan after this was that the team would use January for user testing, bugfixing, training and integration work, before integrating the system into production February 1. Before sprint 5, the plan was to introduce the system 1. January, but this was as previously noted, pushed back a month. Before the post-game phase, in late December, the team was not looking forward to this period, fearing that this would be a hectic period requiring a lot of work.

When the preliminary findings of the study were presented, the Scrum master revealed that the systems introduction was pushed back another month to March. This was credited partly to an extended sick-leave, but probably also because the team needed more time to do system testing.

4.3 Applying Scrum

Scrum has quite a bit of wiggle room on how to actually implement the practices. In addition, it is never an easy task to comply 100% when a new process is introduced, and it might not be beneficial to do so.

This section describes how the team implemented and used Scrum in their process through estimation, planning, sprinting and adapting. It also describes benefits and problems in the development phase, and how Scrum played a part in these.

4.3.1 Estimating

When the product owner finished the product backlog in the pre-game phase, the team had an estimation meeting where all the items in the product backlog were estimated. This was a relatively quick process using planning poker[5] which received quite positive feedback by the team members. Two of the developers, the Scrum master and one of the researchers attended this

[5]Planning poker is a wide band delphi technique, using a special card deck as the medium. Described in Haugen [2006]

Figure 4.12: Estimation meeting prior to sprint 3

meeting. Later in the project, the team had additional estimation meetings where the initial estimates were estimated.

Planning a backlog item, used the following process: The Scrum master selected an item from the product backlog which was then considered by the estimators. Using the cards from the Fibonacci based card deck, the participants would show their estimate at the same time. Sometimes the team would discuss their numbers and re-estimate, but usually the median value of the card values was used as final estimate. The estimates were logged in a spreadsheet, and put into the product backlog.

The estimation unit used was hours. The estimates was often compared with the number of logged hours, to get a feeling for the estimation accuracy. As seen in Appendix A, this was on average through the entire project about 90%, meaning that the estimates usually was a little bit higher than time reported. The estimates was frequently referred to later, and they were especially important in review meetings (see section 4.3.3).

4.3.2 Planning

There was relatively little architectural or high-level planning done.

Id	Backlog Item Name	State	Reason	Sprint	Estimated Effort	Work Remaining
704	Innlogging (inkl AD)	In Progress	Work has commenced	10	24	9
706	Konfigurer menyliste	Done	Work has been completed	4	40	0
707	Sortering	Done	Work has been completed	3	2	0
708	Filtrering Henvendelse	In Progress	Work has commenced	9	8	6
709	Kolonnevisning Henvendelse	Done	Work has been completed	3	Not Entered	0
710	Valg av kolonner som skal vises	Done	Work has been completed	3	16	0
711	Antall Henvendelser	Not Done	New	9	1	2
712	Antall faxer Kundefax	Not Done	New	9	1	2
713	Antall faxer tilbakemeldinger	Not Done	New	9	1	2
714	Antall Kundemailer	Not Done	New	9	1	3
715	Antall Tilbakemeldingsmailer	Not Done	New	9	3	3

Figure 4.13: Part of the product backlog shown in VSTS

The team did some work in pre-game phase through meetings where the arhictectural design was discussed. They also addressed data design, the use of architectural patterns, and the main concerns for the system. These were however used as a starting point, and did in not remain set in stone through the project.

Towards the end of the pre-game phase the Scrum master and product owner created a rough plan of the later sprints, where areas of desired functionality was distributed in the available time. There was no planning of dependencies, milestones or freeze-points.

There were two artifacts important in the long and short term planning process. Namely the product and the sprint backlogs:

The Product Backlog

The product backlog was the starting point for the development phase, and work on this started in the design-sprint after the interface design was finished. These interface sketches, combined with the existing system was the basis for creating the product backlog. This becomes quite clear when regarding the finished backlog, which was focused towards the graphical user interface[6]. Scope of these items varied with regard to implementation from

[6] GUI - Graphical User Interface

trivial to potentially very complex. Sometimes an item could be a simple task such as adding a textbox to modify an attribute:

> Add request description

Other items required significant work on an several levels of the system:

> Search for customer using name, phone number, fax

Also, the finished backlog had according to the team some shortcomings with regard to describing the system as a whole, as it focused on the changes and improvements to the existing systems. In some areas the description was unspecific and ambiguous. Other items were large pieces of functionality, which ideally should have been divided into smaller ones, and which were estimated to require more than a weeks work.

When developers were unsure of what the these items actually required, they would usually discuss it with the product owner. However, at some occasions the product owner wanted to create specification documents for these modules instead of putting the desired features in the product backlog. The following conversation from the review meeting after sprint 6, about uncertainties in a map module, illustrates this.

> Product Owner: What kind of specification document do we have on maps?
>
> Developer: It is kind of vague.

After some discussion the product owner wanted to create a specification document for this subsystem. This was opposed by the developers, who wanted a prioritized backlog.

> Scrum master: The customer needs to make a specification document, however the customer does not need to create a specification document which is 50 pages long and detailed beyond insanity.
>
> Product Owner: Start working on it, and if it proves to require endless work we can discuss it.

This is perhaps a reflection of this being a re-engineering project, but it also shows that old habits die slowly.

The product backlog did change somewhat in the project, but generally there were few features added to the actual backlog. The meaning, content and understanding of backlog items did however change, but this was not reflected in the name of product backlog items.

Instead, the developers would usually add the new information to the items comments-section in VSTS (Figure 4.18 shows a screenshot of this). If the

changes were not recorded there, the developer responsible used printouts of e-mails or other documentation and kept these on the desk. An item name could as a consequence of this remain quite different than the actual feature, as the description and content changed with communication between developers and product owner.

Despite the shortcomings described, the team was generally satisfied with how the product backlog worked in practice. When something was unclear, the team was quick to ask the product owner for clarification, and this did probably fill most of the gaps in clarity.

Creating a sprint backlog

Before each sprint, a sprint backlog was created which in essence was a subset of the product backlog. The product backlog was prioritized on the different "areas", and rarely on actual product backlog items. An area would encompass several related items from the product backlog. This practice was possibly a natural consequence of the sometimes very fine grained product backlog.

Common in all the everyday sprints was that the sprint backlog was filled with tasks which had a combined estimated effort far exceeding the hours available. On average, the sprint backlog contained 677 estimated hours, peaking in sprint 4 with 794 hours. When compared with the average number available (345 hours) and used (287 hours), unsurprisingly none of these sprint backlogs were completed.

The team tried several times to limit the number of hours. An example of this is when planning sprint 3. After planning this sprint they landed on approximately 300 hours which would have been possible. However after unfinished and unstarted items from sprint 2 were transferred, as well as additional items, the estimated total was 624 hours.

Conversations with the Scrum master suggests that this was a conscious decision in the beginning, and was sometimes related to tasks that where dependant on external issues, such as training. The basic idea was to not run out of things to do. Instead of adding items if the team finished the sprint backlog, the backlog was filled so the team always had work.

A consequence of this practice was that the team would have many unstarted or unfinished tasks at the end of a sprint. These were more or less automatically transferred to the next. When creating the next sprint backlog, they would not take into consideration the tasks inherited from last sprint, leaving the sprint backlog overfilled yet again. This became a problematic issue after a while, because the team would create the sprint backlog before considering

the remaining work from the last sprint.

> Scrum Master: I don't want this many items under started. We have been burned by that before. Then it is better to move things away from here.

As the above quote illustrate, even when trying to limit the size of the sprint backlog, unfinished tasks would remain. This issue was not resolved before the emergency period, when the 14 day sprints made it easier to size the backlogs.

The sprint backlogs were usually composed out of tasks that were copied directly from the parent product backlog item. It also contained orphan tasks that were unrelated to an actual item in the product backlog, but still was necessary. An example orphan task from Sprint 7:

> Separate test from development environment.

The close link between product items and sprint tasks, can be illustrated by the average break down factor. When disregarding orphan items, a product backlog item was broken down to an average of 1.17 tasks.

One thing commented by team members in the interviews, and confirmed by the sprint backlogs, was that some sprint tasks were part of several sprints. As an example, one product backlog item was present in all the sprint backlogs from sprint 1 to 8, except sprint 6.

The reasons for this was also discussed. The prevailing argument given was that the developers was afraid of starting on that item/area, because they felt that they didn't have the required experience to handle it. The item was then put in the sprints but postponed until it was necessary to do it, even if the product owner had prioritized this functionality. In the end, the developers gained the necessary experience on this area through other projects and felt confident enough to start on it.

4.3.3 Sprinting

In the everyday phase the team used 30 day sprints which changed to 14 day sprints in the emergency phase.

The team could work quite independent, and was free to choose the appropriate technical solutions. One that presented itself in sprint 3 was that the database layer had been created in a less than optimal way, which made further development more difficult than necessary. This was during the summer months, when most of the team was on vacation. However, when the remaining developer identified this as a problem, he found and successfully implemented a better solution after quickly discussing it with those present.

This is a good example of the prevailing work situation in the sprints, illustrating that discussing and then implementing was the norm when faced with challenges.

Officially, a sprint started the day after a review meeting, but the team usually had a few days before doing sprint planning and really getting started. This time usually used to "clean up" after the last sprint, to fix small problems which surfaced in the review meetings.

The work in the sprints was controlled by the team using the sprint backlog. Task distribution was usually discussed in the planning meetings, and decided by the team. The Scrum master stated that he would sometimes give suggestions on who worked on the different tasks, but the team had final word, and was quite happy with this independence.

One incident related to task distribution happened in December. In the interviews, the team members were asked whether or not they could continue another members work if this member was hit and injured by the Gråkallen tram. All of the team members responded more or less positively on this question. Tragicomically, about one week after the interviews, the main developer was injured in an accident and became absent on sick leave for the remainder of the year. Incidentally this gave the remaining developers a chance to prove what they had earlier said. After a discussion with the product owner they decided to postpone the functionality which the unfortunate developer worked on, focusing on the modules prioritized.

Use Of The Sprint Backlog

The sprint backlog was used in all of the sprints and was a guide to the day-to-day work situation. The developers planned the sprints based on the sprint backlog, assigned tasks and used it to control what was worked on.

The sometimes very detailed product backlog and the 1 to 1 mapping had one implication which was mentioned by the team. When doing small tasks it was common to work in parallel on the ones that were related. This was in itself not a problem, but was said to influence how they wrote down tasks.

As discussed in section 4.3.2, a recurring problem in the everyday sprints was huge sprint backlogs. This influenced the team's commitment to finish a sprint. As they knew that a backlog was too large and impossible finish, the commitment to do so became low. It decreased the urgency to finish. One developer explains this in the following quote:

> Developer: [...] I think that as time in a sprint passed, and we realized that we wouldn't finish all the tasks anyhow, it didn't matter that much whether the sprint was 70% or 90% done.

The low importance of the burndown chart discussed in section 4.3.3 can be regarded as one symptom of this. As the team knew they would never reach zero anyway, and by this regarded the burndown as more or less useless.

Another symptom was that the developers felt that this situation made it difficult to maintain a complete overview of the sprint:

> Researcher: Do you from sprint to sprint maintain a clear image of what is supposed to be delivered?
>
> Developer: No, not in the firsts sprints. But there is more focus now on what we wish to deliver in the sprint.
>
> Researcher: When did you gain this overview?
>
> Developer: After sprint 3 maybe?

Meetings

The meetings are an important part of Scrum, and was also an important part of this project. While the use and importance of daily Scrum meetings varied, they still gave important benefits. The review meetings were regarded as the most important ones, especially as an opportunity to get feedback and explanation on functionality.

Daily Scrum meetings The Scrum meetings were quite informal and unscripted through the entire project, but many of the other properties changed quite a bit from the everyday to the emergency phase. In the everyday phase, meetings would take place about 9 (often later) in the main conference room (see figure 4.6). The team would sit down, have coffee and talk about sports or other events for a short while before starting the actual meeting. It was then usual to address and discuss project issues, often quite detailed. Due to this, the meetings were often quite long (15+ minutes). Sometimes in the beginning of the project, the Scrum master showed the burndown chart, but this practice vanished altogether after a while. The meetings were not strictly daily, and when the Scrum master was absent, the team would rarely have daily meetings by themselves.

There can be several explanations to why the meetings had this form. One explanation (a recurring one, which affected many issues in this project) is the limited time available to some the developers, which probably increased the need to discuss technical issues in the Scrum meetings. Another explanation is that the venue was a meeting room, far away from the "home ground" where the developers had their things, including the index card wall. Sitting down in a meeting room probably invited to more discussion,

Figure 4.14: Cartoons next to the index card wall, with a demotivational poster[a] on the virtues of meetings.

[a] A demotivational poster is a parody to the more usual inspirational or motivational posters. As an example, a motivational version of "Meetings" could advertise meetings as "Two minds think better than one", while the demotivational version could be "None of us is as dumb as all of us".

as in a proper meeting, instead of the short update-session indented with a traditional scrum meeting.

Regardless of the format, the daily Scrum meetings was still valuable, probably especially so for the junior developer. The technical discussions helped the team maintaining a team effort, which probably would have suffered without these meetings. In the sometimes chaotic situation with multiple projects, priorities and interruptions, daily meetings in some ways ensured team communication. A developer expressed this in the following quote:

> Developer: I think we had a positive sprint. If we hadn't used Scrum we would still be messing about on our individual computers. [...] We would have lost focus on the project as a whole.

In the emergency phase, a lot of things changed, and also Scrum the meetings was tweaked quite a bit. The Scrum meetings was now held while standing/sitting in front of the index card wall close to the work area. When the Scrummaster was absent, the developers sometimes had a short update-session by themselves in the beginning of the day. The duration was shorter and there was less technical discussion, suggesting that this was not needed to the same extent as before as the team members cooperated more of the time. Although our data from this period is less complete than from the everyday period, observations and the interviews back up this tendency.

Review meetings The review meetings was according to some of the team members one of the most successfull aspects of the Scrum in the project.

The review meetings generally had the folllowing pattern: Before the meeting started, the team did last minute adjustments, bug fixing and wrote down tasks which could be demoed. The product owners travelled by plane and was on site for the meeting which would usually start at around 10. If they could not attend in person, teleconference and desktop sharing was used. Review meetings were usually around 4 hours, somewhat shorter in the emergency phase.

When the meeting started, the Scrum master would first describe the general status of the project; hours used, hours written down, the important impediments, reasons for few lacking progress and so on. It was quite usual to have a longer discussion on these things before proceeding. The following conversation is from one of the earlier review meetings:

> Scrum master: After last sprint we created this list and we all agreed on it. We had inserted too many hours, Greg disappeared on paternity leave before expected because of his wife went early into labour, and Howard did not enter the project as planned.

Figure 4.15: Daily Scrum meeting in the emergency phase

Scrum master: 24 of May we had 600 hours remaining, now we have 300 hours.

Product owner 1: How can you say that this is good?

Scrum master: If we had the resources we planned when creating the sprint, things would have been OK.

Product owner 2: OK. The most important thing is that the paternity leave came now.

Developer: This can't be 300 hours.

Scrum master: No, but we started with to many hours. Next time we need to start on the right level compared with resources.

Product owner 1: How much of available resources used on the project were used compared with the planned? This says nothing of what has you have actually done.

Scrum master: I will make a summary of hours used, and then we can look at how this compares with hours written down. Then we can consider the estimates.

As seen in the above quote, hours were a discussed topic. The tendency was to discuss hours used and written down in the beginning of the review meetings, and this persisted through the entire project except from sprint 1. In these sprints, the hours were used more as a progress measure than in the beginning. The following quote is from a review meeting in the emergency phase:

> Product owner 2: Good! You have written down more hours than you have used.

The above quotes are quite typical for the review meetings, also in the emergency phase. Hours written down and used were important when measuring the success of the sprints. Sprints 3 and 4 were regarded as the most successfull sprints in the everyday period, and these had a higher percentage of hours used than sprints 1,2 and 5.

Worth to note was that team generally perceived the product owners as positive after the review meetings.

> Scrum master: Review was positive as usual.

The next issue on the agenda was demonstration of the functionality produced. The developer responsible for an area of functionality would do this demonstration, and the product owner would comment as the demo progressed. After the demo, the participants would discuss adjustments and technical details around the functionality produced.

The last item on the schedule was discussions on the next sprint. Here the Scrum master, developers and product owner prioritized the areas for the next sprint, as discussed in 4.3.2. After this, the sprint was essentially planned, and the review meeting was concluded.

Planning meetings As the description of the review meeting shows, the size of a sprint backlog was essentially determined in the review meeting. The team still had planning meetings afterwards, in meetings which usually were around 2 hours long. This was done before all of the sprints. The agenda was on discussing the progress of the sprint, add orphan items and discuss possible technical issues. Re-estimation of product backlog items was also done in these meetings.

Retrospective meetings Only one retrospective meeting was performed in the project. This was held in week 35 after sprint 3. Quite a few issues were addressed in this meeting. As figure 4.16 shows, the team used post-it notes to identify positives and negatives in the project, as well as possible improvements. The team members first created notes on positive and neg-

Figure 4.16: Retrospective meeting held after sprint 3.

atives in the project and grouped the notes together depending on topic. After a discussion, the topics were ranked according to priority. The team then created new notes with possible solutions to these.

The following list is a summary of the results of the retrospective.

- Dependencies on other projects
- Resources
  - Hiring new people
  - Prioritize projects
  - Other available developers
- Testing
  - Create a check-list for testing
  - Set a deadline/code freeze some time before a sprint is completed
- Discuss technical choices at daily Scrum meeting
- Meeting/Process/Tasks

- Developer 1 is responsible for the finished product
- Daily meetings regardless of who is present
- Do something with the breaking down of product backlog items

As the project progressed, some of these issues were addressed. The majority of these changes happened first in the emergency phase.

Finished! (for testing)

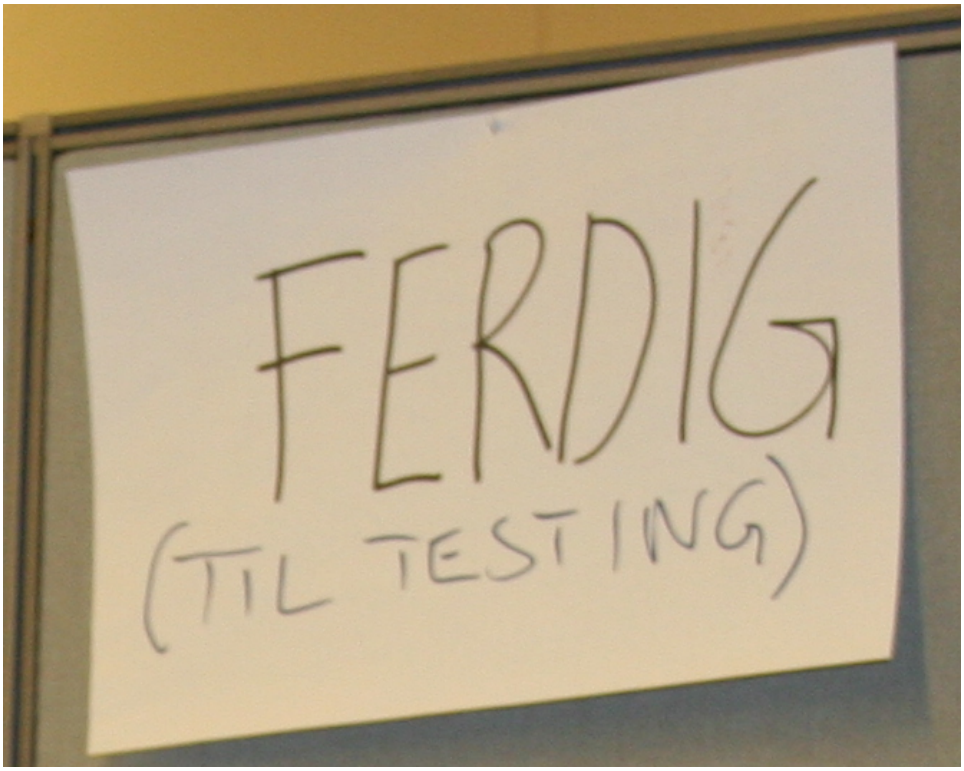


Figure 4.17: Finished-section of the index card wall, week 45.

A important principle in Scrum is the empirical process management, where the state of a process is constantly monitored and adjusted. The sprint backlog coupled with the release and product backlog is the one of the two process management techniques proposed by Scrum project (the other is first hand observations [Schwaber, Chapter 4.3]). One issue related to this but not covered in detail by Scrum literature is when to declare a sprint task, or a product backlog item as done.

The criteria for finishing a sprint task (and in most cases, also a product backlog item), was not explicitly defined in this project. Generally, a task

was declared as finished when the developer responsible declared it so. Figure 4.17 shows the label for the done-section of the index-card wall, which reads "Finished (for testing)". The criteria "for testing", was added quite late in the project (after week 40), and meant that the developer deemed it ready for user testing. Organized system user testing began first in week 51, and feedback from this could naturally not be incorporated into the earlier sprints.

The relatively undefined finished-criteria had a subtle effected on progress measuring. The team naturally wanted to finish functionality, and especially so before a review meeting, in order to show progress to the product owners. This sometimes meant that functionality that was mostly done, but perhaps had a few known bugs, was declared as finished even if the developer responsible knew that some work remained. It was not that uncommon that functionality being demoed in review meetings crashed, or functioned worse than hoped for. Usually, some of the remaining work was then done in the beginning of the next sprint, which can explain why there was little progress in the starting period of a sprint (as described earlier in this section). Our data also suggests that this tendency was stronger in the emergency phase when the pressure to deliver functionality was more acute.

Tool Support

The Scrum process was supported by both computerized and manual tools.

The main computerized tool suite was Microsoft's Visual Studio Team System (VSTS). This integrated most aspects of the development process such as coding, testing, building and bug tracking with process support. Although Scrum was unsupported out-of-the-box, a 3rd party plug-in provided support for most Scrum-related aspects in the beginning, with an official plug in released in sprint 3.

The product backlog was tracked with description, area, estimates and additional information of the backlog items (See figures 4.18 and 4.13 for examples). Additional information relating to the product backlog features was added continuously by the developers, for instance after receiving clarification on a issue from the product owner. While this was not done consistently, the possibility to do so was popular with the team. The product backlog items were also linked to sprints, and with the related sprint tasks.

Sprint backlog tasks were tracked in much of the same way as product backlog items. Checked in code could also be linked to a sprint task, but this feature was not used. As developers finished tasks, they would write down the estimated time left. The burndown chart was a product of the sprint backlog

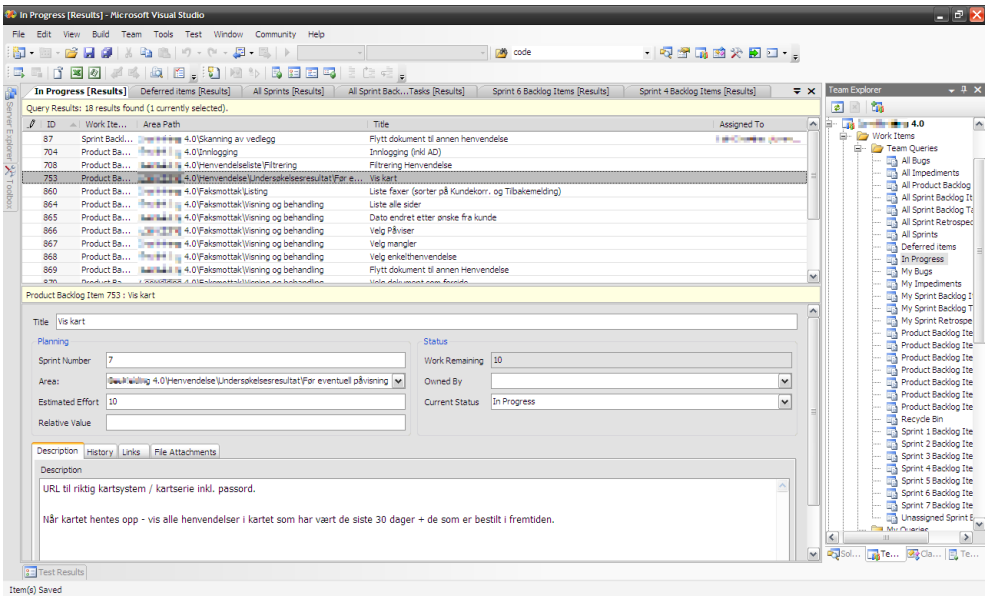


Figure 4.18: Screenshot from Visual Studio of product backlog showing items in progress

tracking, and was by this always up to date and available in VSTS or in a project web portal where project documents could be accessed.

At the start of sprint 3 one of the researchers introduced index cards, and helped create these initially and later in the project, based on the sprint backlog. These were then put on a wall in the development area next to the developers island. As figure 4.19 show, the wall was divided into three segments: “not started”, “in progress“, and ”finished“. The wall was well received by the team and especially by the Scrum master:

> Scrum Master: I have got a REALLY good start on your cards. I think it will be a very good thing to have this visually on the wall. It will be interesting to follow up on estimates.

The index card wall was after this a very visible part of the project, but did not replace the computerized system, and the two systems were used in parallel. Developers would usually update the estimated number of hours left on the computer, while the index cards were moved from the different segments of the wall as tasks were started or finished. One of the developers said that he would move an index card from left to right in the “in progress” segment as the corresponding task progressed. A newly started task would be on the border to “started”, while one that neared finish in the right of the segment close to the border of “finished”.

Figure 4.19: The index card wall in week 35.

As previously mentioned, the burndown chart was seldom used in the daily scrum meetings. In large parts of the project it was only visible when the Scrum Master showed it or when a team accessed it on the computer. This changed for a while after the introduction of the index card wall, when a printout of the chart was put on the wall next to the index cards. Although this was maintained in Team System, and available through the development environment and in the projects web portal, it was not used to any extent. The printouts were relatively infrequently updated.

In sprint 6, early in the emergency Scrum phase, the printout of the burndown chart was for a short time replaced with a manual version shown in figure 4.21. This was at the same time as the team started having the daily Scrum meeting next to the index card wall. According to one of the developers, this was to improve visibility of the burndown, but the effort was relatively short lived. The burndown chart was not used to any extent in the last two sprints.

Figure 4.20: Index cards. The information is; Task name, Task area, initial hour, remaining hours.

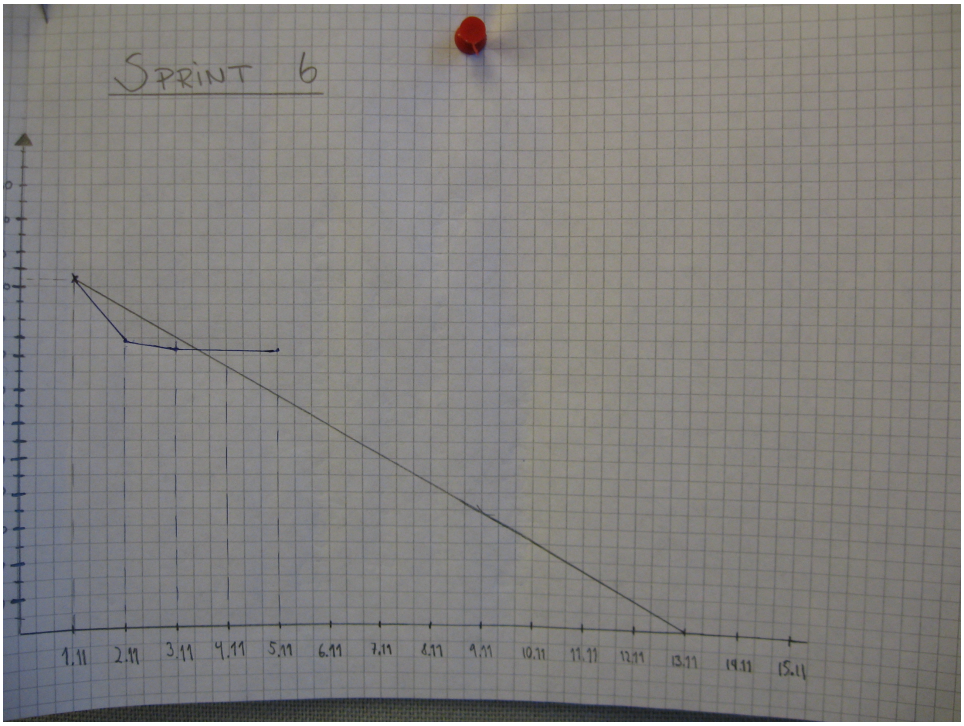


Figure 4.21: Manually calculated burndown chart introduced in sprint 6

Closing The Sprint: Measuring Progress

As described in section 4.3.3, an important progress measure in the review meetings was hours used and not functionality delivered. This did not mean that the delivered functionality was not important. After sprint 1 the team had produced a subsystem which was introduced in two of the regional offices, and was according to the product owner well received by the operators there. In sprint 2 and 3 the team created more back-end systems which was not as visible. In the later sprints, the team worked on creating the rest of the system and had more "demo-friendly" things to show in the review meetings.

As the sprints were seldom finished, fulfilling the planned deliveries of a sprint was not that important. This was reflected in the team through various remarks and statements in the interviews. Also the product owner was seldom very concerned in the review meetings when the sprint backlog was unfinished. The emphasis was more on the actual number of hours used, than the functionality delivered. This is perhaps not that surprising, given that the project planned to integrate the system in one big bang rather than doing it continuously. The whole system in the end was more important then 3 modules delivered in a sprint, and a missed delivery was a regarded as a slip and not a tendency.

4.3.4 Adapting

As described in section 4.2.2, the project fell behind schedule about mid-way. The team probably knew this quite early, but did not introduced measures to improve progress before sprint 5 was finished.

That the team had an idea that things weren't going quite as planned, was highlighted by several remarks along the way as well as in the interviews. The lack of resources was an identified concern from the start, and was also discussed in the retrospective meeting. After the sprint 5 review meeting one of the developers had the following comment:

> Developer: It is strange that it's a surprise to the others that we won't make the deadline. 3000 hours were allocated, but only 1500 have been available.

The large sprint backlogs and tendency of many unfinished and ongoing tasks at the end of a sprint probably concealed the actual progress quite a bit, and contributed to the late discovery of the problems meeting the deadline. A generally optimistic approach to planning also contributed to the late discovery, as the team tended to generalize on the best-case sprints (sprints 3 and 4).

The team, and especially the Scrum master probably needed concrete evidence to justify drastic changes to the process. As these changes were both internal (shorter sprints, overtime, etc.) and external to the project, justification was clearly required by convince management. Some self-justification was probably also needed.

When sprint 5 didn't go as planned, the team was forced to make changes. The Scrum master notified the product owners in the middle of the sprint, and decided on measures in the following review meeting. These measures are described in the last part of section 4.2.2. These changes were visible on several levels; The backlogs were generally more adjusted to available resources, but were still overfilled to some extent. As the timespan was shorter, the team had a better understanding of the whole sprint. While the team didn't meet the delivery plan all the time, they were generally much closer to the target. This, combined with increased team protection and more time improved sprint commitment considerably. The product owners were still quite concerned with the number of hours used, perhaps out of habit.

In the emergency phase, the team and product responded by increasing both the effort (team protection, more time) and control measures. Due to the narrow time frame left before the deadline, the increased control was probably needed, as further slips in the planned progress would have been

critical.

The sprints in Scrum enable and encourage continuous integration and testing. These possibilities were only partly utilized, as described in section 4.1.2. As the study ended before organized full scale system testing had started, we have little data on the actual build quality of the system. Based on the fact that the deadline was postponed two times, it is probable that this would have benefited from more dedicated testing after sprint releases. The lack of experienced testers in the team could also explain this.

Chapter 5

Results

In the previous chapter, some of inner workings of the case was described. Hopefully, the reader gained an understanding of the ups and downs of the project, their challenges, possibilities and solutions.

In this chapter, the research questions will be discussed further. To repeat, the research questions were the following:

> RQ 1: How is Scrum implemented in a project compared to the book? What are the benefits and disadvantages?
>
> RQ 2: How does Scrum support teamwork?

The two first sections in this chapter are dedicated to each of the research questions. The third addressed the validity of the study.

5.1 Scrum'ed

In the course of the project, many of the things relating to Scrum varied in such a manner that it is sometimes tempting to regard the project as two different cases (see section 4.2 for a overview of this). This discussion will focus on the most visible themes in the project, and consider eventual change in this when relevant for confirmation.

In the following sections, the compliance of the project studied is discussed. Compliance is defined as follows:

> S: (n) conformity, conformation, compliance, abidance (acting according to certain accepted standards) WordNet

The term compliance when used in an Agile setting is perhaps a poor choice of words, because Agility as described in the Agile manifesto [Beck et al.,

2001] supposed to be "Individuals and interactions over processes and tools". The motivation for this discussion is inspired by this, not to cast any form of judgment, but rather highlight and discuss how differences in the implementation from the book influenced the project.

5.1.1 Compliant

Many of the most visible parts of the project was well within the descriptions in literature. This sections discuss the Scrum-compliant practices, and how they attributed to the project.

The following items were considered to be compliant within the bounds described in literature.

- Scrum fundamentals; sprints, roles and co-location.
- Product is planned through product and sprint backlogs
- Communication with Product owner through backlogs, review meetings and frequent communication
- Team was trusted to work independently
- Adapted to technical challenges and opportunities
- Estimation
- Progress issues were discovered and handled

The team used 30 day sprints in the first period of the project, which was changed to 14 day sprints after recognizing that the project was behind schedule. Within the sprints they worked on tasks from a sprint backlog, and conducted (mostly) daily Scrum meetings. When finishing of sprints, the team and product owners performed sprint review meetings, where the finished functionality was demonstrated to the product owners. At the end of this meeting, the team got instructions on what was prioritized, and created a new sprint backlog based on this. This corresponds well with the description in theory.

Planning and specifications is in Scrum mainly done through the product backlogs. Existing theory is not specific on how a product backlog is supposed to be, all that is said is that it is supposed to be a prioritized list of business and technical functionality [Schwaber and Beedle, 2001]. While the product backlog in this project had it's issues, as described in section 4.3.2, especially on the granularity of tasks and a strong GUI-focus, these proved to be less important. Mostly because of good communication between the product owner and the team.

Most aspects of the customer side of the project functioned from the authors perspective well, and was mostly within the bounds of Scrum. Albeit Scrum specifies one, and there initially were two product owners, these adopted separate functions, with one acting as a reference person (product owner 1), and one as a product manager (product owner 2). Similar to the description in section 2.5.1, the product owner in cooperation with the Scrum master created and managed the product backlog and was the final authority on questions and priorities related to this.

Review meetings were an especially important practice in communication, and enabled discussions between product owners and the entire team. They presented, according to the team, an opportunity to "shed light" on issues. This description correlates with the findings in Mann and Maurer [2005]. It could be argued that an on-site product owner would have been preferential, but this possibility was discussed early, and rejected because of a lack of a suitable person. The early GUI-work, the review meetings and communication through telecommunication was probably sufficient, and an added benefit was that the product owner had a good understanding of the system as a whole.

The team was trusted to distribute tasks and work independently pretty much as described (see section 2.5.1. This part is challenging for a Scrum master, but he did mostly stay out of the task distribution process. At the most, there could be suggestions when a sprint backlog was divided, but the team always had the final word and would juggle tasks as they saw fit. Also in the technical aspects, the team was very independent and could incorporate other projects, open-source modules and generally create the system as they saw fit. Schwaber and Beedle [2001] writes that a team should be granted full authority to do whatever is necessary to fullfill a sprint, and in the technical aspects this is a good description of how things were handled in this project.

Estimation in Scrum is based on the notion that estimates are unreliable, that they probably will improve with time and experience, that the people actually doing the tasks are best suited to estimate, and that they are not binding [Schwaber and Beedle, 2001]. This description fits well with the estimation process used. The team used planning poker to estimate the product backlog, a process which was surprisingly accurate. The estimates was then used to size the sprints. While the size of sprints was an issue, the estimation process was not the cause of this.

The metamorphosis of the project from the everyday to the emergency phase, illustrated that the team could adapt effectively based on the information and progress measures acquired through the sprints. This is consistent with the description of empirical process control described in theory.

5.1.2 Deviations And Adaptations

The practices in this section are considered to deviate from the by-the-book way of doing them. However, they worked reasonably well and are regarded as adaptations to the process.

- Much discussion in daily Scrum meetings
- No use of the burndown chart
- Little or no breaking down of items when planning sprints
- Long pre-game phase

As described in section 4.3.3 the setting of the meeting differed in the every-day phase somewhat from description in Schwaber and Beedle [2001]. This was also the case with the focus, which was significantly more discussions on details. The meetings were seldom only a status update. This is similar to Mann and Maurer [2005] where the daily meetings were found useful, but had a tendency to drag on if the team was unfocused. This is considered an adaptation because these meetings still was a valuable contribution to the project, and increased communication between team members that would otherwise be working more separately. In the emergency phase, the daily meetings changed, and became more like the short, to the point meetings described in literature. The increased team effort, focus and communication probably decreased the need to discuss, because issues had been sorted prior to the meeting. Short, focused meetings, was in this study an indication of more effective teamwork.

The burndown chart is a quite visible and advertised artifact in Scrum (there are several sample burndown charts in theory [Schwaber and Beedle, 2001, Schwaber]. It is described as a tool for visualizing remaining work and progress, and is a derivative of the sprint backlog. It is interesting that it was not used seriously in any part of the project, and was never found valuable. There are several practices that contributed to this. The most important one was too large sprint backlogs, and tendency of not aiming at or reaching zero hours left. Another one, was the relative invisibility of the burndown in the first sprints when only the computerized system supported Scrum.

As these practices changed in the emergency phase, the team did not successfully re-introduce the burndown. The index card wall was instead the primary sprint progress indicator, with the adaptation of moving the cards from left to right as tasks progressed. This is somewhat similar to the situation described in Fitzgerald et al. [2006]. The absence and low importance of the burndown chart was probably not important to the overall efforts in the project. However it could be regarded as a symptom of low sprint

commitment.

When creating a sprint backlogs, theory suggests that the team should transform the goals and items from the product backlog into the discrete tasks necessary to reach the goal [Schwaber and Beedle, 2001]. This was rarely the case, as the average break-down rate of 1.17 tasks per items confirm. One reason for this might be the sometimes very fine-grained product backlog, which in most cases made it unnecessary to break down items. All in all this probably did not affect the project that much, however one developer felt that it could have beneficial to do it to a larger extent. Similar to this situation was the product backlog in Lervåg [2006] which was found to be highly technical and detailed.

The pre-game phase of this project was relatively long. Production of GUI-sketches was a main part of this period, as well as some high-level planning. Scrum allows a pre-game phase, but states that it should be as short as possible. Because this period created a reference that supported later work instead of inhibiting it, I think it was a valuable adaptation to the process. Other studies on Scrum-projects [Cloke, 2007, Fitzgerald et al., 2006] also refers to a sprint 0 or a planning phase where the further sprints are planned, showing that this is not unusual.

5.1.3 The Challenges

All project have aspects that works great, aspects that works, and aspects that needs changing. This section describe the aspects that deviated much from Scrum theory or studies and which plainly didn't work. They were absent or implemented in such a way that they became a liability to the project. Together, they formed a whole that had negative effects on the project performance and monitoring of progress. The list of suspects are:

- Interference, intruders, but no peddlers
- Creation of sprint backlog before discussion
- Consistently large sprint backlogs
- Little or no commitment to sprints
- Hours as a progress measure
- Potentially shippable?
- Not so steady deliveries

The first item in the list of challenges is a direct reference to chapter 3.6.2 in Schwaber and Beedle [2001], which is named "No Interference, No Intruders, No Peddlers". Scrum preaches highly autonomous teams that controls their

own work day. While the team was generally independent internally in the project, it was a completely different story when considering the work situation as a whole. Customer support, other projects and disturbances forced two of the team members to divide their efforts, and decreased the priority on the project. This degraded the alignment towards the team, which was supported by statements from developers. It had ripple effects that are important when trying to understand a number of other issues. That it was considered to relocate the team to separate offices, is illustrative of the level of interruptions. The experiences from the emergency phase, when there was more effective team protection reinforces the view that this was a significant problem.

The process used when creating the sprint backlogs deviated only slightly from theory, but can still be regarded as an important prerequisite to the problems with sprint size. As described in section 2.5.1, Scrum states that the product owner should only prioritize the product backlog, and later cooperate with the team to create the sprint backlog in a later sprint planning meeting. Generally the sprint backlog was calculated and created in the last part of review meetings, and discussed afterwards in an extended planning meeting. In other words, the scope of the sprint was essentially decided, before the team could discuss what they could realistically do. Why this became the norm is difficult to say. Possibly, it was the convenience of doing setting the sprint when the product owners were on site that led to it being the usual way to do it.

The sprint backlogs were large, often twice the size of estimated team capacity. Theory suggests that the team should adjust the sprint backlog according to estimated capacity and previous velocity. As described in section 4.3.2, this description could not be used about the everyday phase of this project. While the initial size often was within scope, the unfinished sprint tasks were automatically transferred from the previous sprint. This created a "shadow backlog" that followed the sprints like a bad habit until sprint 6. There are various explanations. One was the stated motivation to have enough work. Another was an optimistic view of the team capacity, and tendency to generalize on the best-case sprints and projected available hours. The team and especially the Scrum master wanted to show that they could "do it" and over-promised. Using hours as a performance measure, and not adjusting is more the traditional way of thinking, rather than Agile.

In Schwaber, it is stated that the team should at the end of each sprint deliver an increment of potentially shippable functionality. Potentially shippable is a slightly ambiguous term, which in this case meant that the code was ready for (system) testing. Because there were no experienced testers in the team as well as a lack of tradition of doing structured tests when developing, this was postponed to after the deadline. This could mean that quality issues

were discovered late, but as our data contain few indications on the build quality it is difficult to say how this turned out. This tendency is similar to the waterfall-model as described in section 2.2.1, where coding is done first with testing done afterwards, and is not compatible with the the concept of delivering "tested" code (section 2.5.1.

A sprint is a time-boxed period of time where the team commits to fullfilling the sprint goal, and is in exchange for commitment given the freedom to do so [Schwaber and Beedle, 2001]. The team never really committed to the sprints. Commitment improved in the emergency phase, but it was still more geared towards the final deadline. The high amount outside interference was arguably the main contributor to this. The large sprint backlogs were also important, both as a cause and an effect. It can be argued that a committed team would never agree to anything but a decently sized backlog. Because of the general interference in the company, this commitment was not there in the beginning, and with the recurring large backlogs, it did not develop further until sprint 6. This type of commitment is again more similar to traditional waterfall where the commitment is towards delivering the final system, rather than delivering increments in the sprints.

One might ask if this all of this really matters, when the premise of Scrum is to deliver steady production ready increments, rate the success of a sprint on the software delivered and regard development as a black box. That question is prudent also in this case. As discussed, the increments were not strictly production ready. Also, the success factor, at least in the review meetings was not only the deliveries but also on hours used compared with hours written down. This shifted some of the focus from finishing sprints to finishing the project before the fixed and much more serious deadline after sprint 9, and was as discussed a waterfall-like part of the project. Why and when the hours became a success factor is uncertain, but our earliest data of it is from the review meeting in sprint 2. It was quite possibly just a defense mechanism to justify not finishing the sprint, or maybe a more convenient way to keep the product owners up to date with progress.

5.2 Teamwork

In section 2.5.3, theory on Scrum was briefly evaluated towards the "Big Five" of teamwork [Salas et al., 2005]. The findings from the study are in this section discussed according to the Big Five. Figure 5.1 represents a simplified graphical representation of this discussion.

As figure 5.1 shows, the ultimate measure of teamwork is effectiveness. This is hard to measure, and also in this study it is difficult to say anything conclusive about without good, quantitative performance measures.

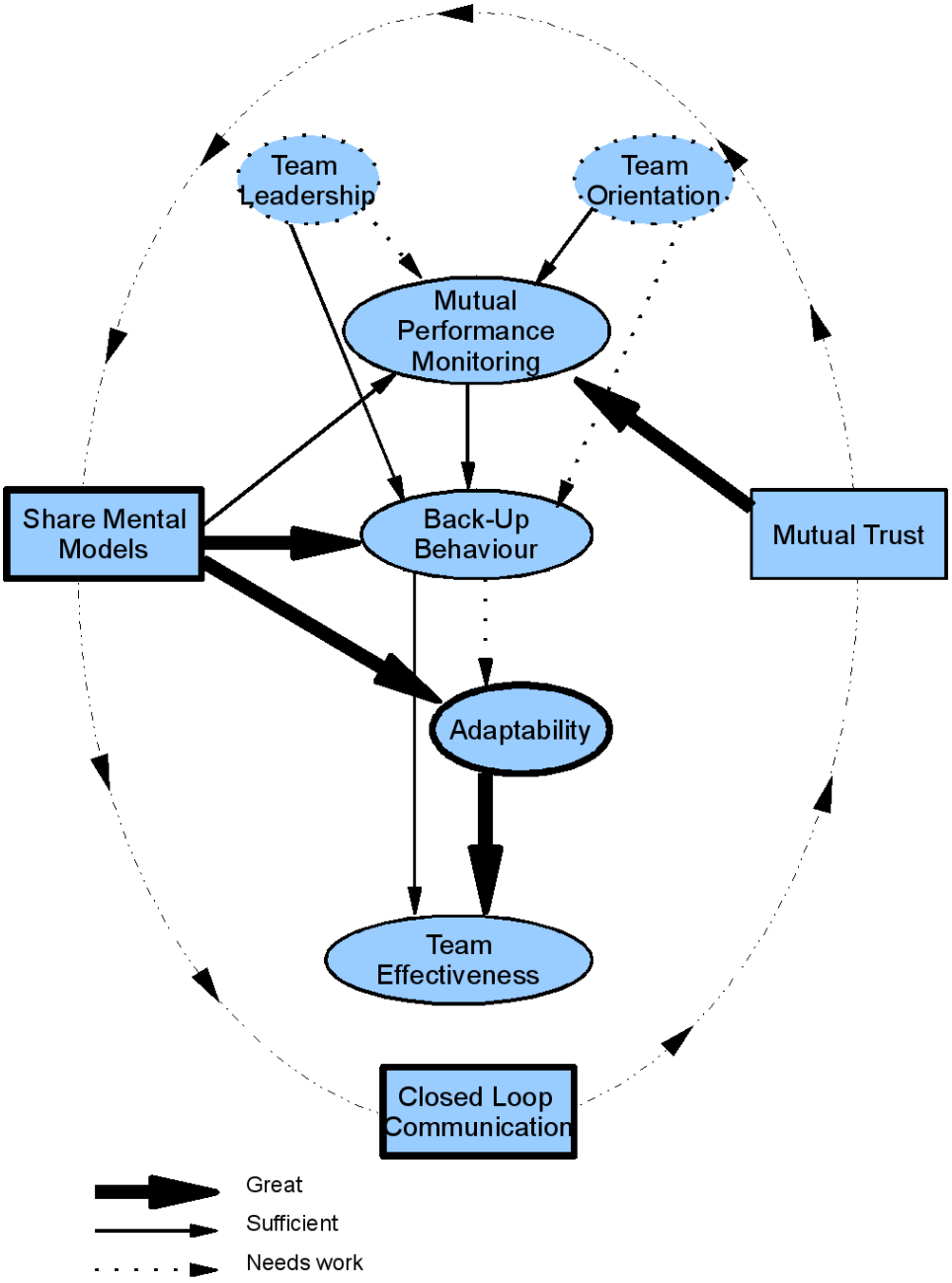


Figure 5.1: Graphical representation of relationships among the Big Five and Coordinating Mechanisms. Adaptation of figure 1 in Salas et al. [2005]

Generally, the data points to an improvement in the teamwork-effectiveness in the emergency phase. Some of the previously weaker aspects was addressed in this phase, and the team worked closer together. This is reinforced by statements made in the interviews.

5.2.1 Coordination Mechanism

Shared Mental Models

The shared mental model is supported by several Scrum practices such as review meetings

By and large, these practices were effectively used throughout the project. In addition to these, the early work on GUI-sketches was beneficial when creating the shared model, especially for the team members involved. Accuracy was reinforced and maintained mainly through the review meetings, but also by communication with the product owner when something was unclear. In the team, members reported that they usually knew what the others were doing.

The work done on GUI sketches in the pre-game phase had a positive effect for establishing a shared model of the system, and was said to have prevented discussions later in the project. They were later used as reference material in the development process. Maintaining this was supported by the co-location of the team. In addition, the estimation meetings was important as they provided an arena where the team had to discuss the entire product backlog. However, perhaps the most important Scrum feature in this respect were the review meetings, where the team could synchronize their mental model with the product owner. When asked about the day to day cooporation with the product owner, one of the developers responded:

> Developer: [...] but it is like, the product owner *knows* the system, so if we wonder about something, we will get a response telling us how things are supposed to be.

As the product owner was in another city, this communication was done using email, phone or teleconference. Review meetings still had a important role:

> Developer: There have been things that required a little more presence, and those have been handled in the sprint review.

The co-location and daily meetings were also important to update members on things happening in other projects.

Closed Loop Communication

In the first two links the communication worked quite well, supported by frequent meetings and co-location. In the everyday-phase, the communication in the daily meetings was a more important than later phases when the team cooperated more. With respect to communication with the product owner, the communication methods available presented a challenge and which at a few incidents lead to misunderstanding. However, given these limitations also this aspect of communication worked and there were few examples of misunderstandings.

Mutual Trust

The team reported that they trusted that the others both could and would do their jobs. One developer stated that there were no cases of other members not taking responsibility. Also technically there seemed to be confidence in the others abilities. Working on shared code segments was much more common than earlier, something which is an indication of mutual trust:

> Developer: I think it is the first time I've experienced this here, that it [code] has been shared this much.

Trust in the team as a whole is a slightly different story. While the findings show that the Scrum master and product owners trusted the team to do their job, they also knew that their job included other projects. The team had clear problems with sprint commitment. This was probaly influenced by knowing the others situation and occasional need to prioritize other tasks.

That the product owner jokingly wished the project was on a fixed-price contract illustrates that they didn't fully trust the team, as they knew they had other and sometimes conflicting responsibilities. Focus on hours as a performance measurement is another indication of this. Being an internal project probably also meant that "real" customers often had priority.

5.2.2 Team Leadership

As described in 2.5.3, leadership in Scrum is a shared responsibility between all of the roles. However, it is an aspect of teamwork which is challenging, especially with regard to the Scrum master which requires a fine balance of intervention and motivation.

Within the team, leadership worked well with relatively few issues to deal with. The team had few problems with self-organization and communicated well with both the Scrum master and product owner.

Externally, the situation was more chaotic due to practices in the company, as described in section 4.1.1. Further, this situation was not helped by the Scrum master having a responsibilities as head of the department. Team protection which is important in Scrum was not implemented effectively before the emergency phase.

The sprint backlogs were in large parts of the project overfilled, and the review meetings used hours as a performance measure. These two practices together created unclear performance expectations, which was reflected in low sprint commitment. This situation had a detrimental effect on mutual performance monitoring and back-up behavior, which is discussed in the next sections.

5.2.3 Mutual Performance Monitoring

Mutual performance monitoring is enabled by daily scrum meetings, and sprint burndown charts on in the sprints, and by review and retrospective meetings in the project.

The daily meetings and the co-location provided an environment where it was easy for the team to discuss issues and keep up to date with what was going on in the project. The communication in the meetings further suggests that the team had the ability to perform mutual performance monitoring, especially in the emergency phase, when they worked together more.

A different interpretation could be that the shorter meetings in the emergency phase meant less cooperation and more individual hacking. This view is however contested by the team members themselves, which was especially pleased with this aspect and how it worked.

While the ability to perform mutual performance monitoring was in place, the unclear performance expectations discussed in section 5.2.2 meant that it wasn't that important to monitor the performance. It wasn't really expected that the team met the goals set in the sprint backlog, and this affected backup behavior negatively because there was little need for it.

5.2.4 Backup Behavior

Backup behavior is about providing feedback, coaching, assistance and completing tasks for each other. As mentioned in section 2.5.3 Scrum has no direct practices to support it, but relies on an effective self organizing team in which this is an important dimension.

The team showed some evidence of effective backup behavior in feedback, coaching and assistance. This was provided mainly by co-location and an

environment of mutual trust. As described in the previous section, there was a lot of communication between the developers when they were working. They described it as effortless to ask for assistance from the other team members if there was a problem. As one of the developers described it:

> Developer: I think it has worked very well sitting together with those involved on the project. When something comes up it has been easy to just ask and get an answer.

Also with the product owner, the team felt that it was quite easy to ask for feedback. Overall, mutual assistance worked well in this project. As with performance monitoring, it seems that this aspects improved in the emergency phase, when the team worked together more of the time.

With regard to feedback and coaching things are more unclear. It is apparent from the data that cooperation was the rule rather than the exception. Sharing of code was quite common, they did on some occasions do pair-programming sessions, and generally had knowledge of how each others code worked (some exceptions probably existed on special modules). With this overlapping knowledge of each others code, they would give feedback when discovering improvements, and coach each others to improve their work.

Whether or not the team completed each others tasks is unknown, and the data points both ways. The answers given in the interviews suggests that some task completion might have happened. The developers said that there were some juggling of tasks going on, that they were quite flexible on who should do a the tasks, and that they sometimes worked on others code. There also are evidence to the contrary. One pattern suggesting this is that when demoing it was usually referred to individual developers when talking about specific functionality. One specific event from week 45 reinforces this view: At this review meeting, the demonstration of a feature was postponed to the next review meeting, because the developer responsible was sick that day. This would be one case where ideally another team member could have done the demo. When it didn't happen then, it implies that is was uncommon also in the rest of the project.

Salas et al. [2005] writes that backup behavior is dependant on both the needs of the team, and the opportunities of the different team members. With this in mind, the relatively low commitment to sprints and the high workload from other projects in the everyday phase inevitably had an influence and degraded the teams backup behavior, they did however help each other. The need was probably also quite low, and as described in the previous section, mutual performance wasn't functioning optimally. When need and opportunity increased in the emergency phase, the effective backup behavior was seen to increase. While task completion probably wasn't done extensively, the assistance, feedback and coaching functioned well.

5.2.5 Adaptability

Increased adaptability is one of the adverticed effects of Agile methods, as a result of the short feedback loops. The team is also able to control their work to a large degree, which further facilitates adaptability.

There were a relatively few technical problems during the project, and the team adapted well to those that arose, and had much liberty in this respect. One example of technical adaptability was when the team discovered and incorporated functionality from another project. Another example was when the plan to use a existing externally developed (and quite expensive) subsystem was dropped because of performance issues, and the team decided to develop it themselves. This type of adaptability was mainly supported by the shared mental model, and good communication.

Adaptability with regard to tasks was also quite effective. Perhaps one of the biggest problems was the limited amount of time available in the everyday phase for 2/3s of the team, and adapting to this was difficult for the team alone. However, the fact that the team could freely distribute tasks on their own helped to avoid bottlenecks, something which was illustrated by one of the developers:

> Developer: At least, it isn't like anything blocks the project if one of us is away. That is a positive thing.

The limited back-up behavior did however mean that there seldom was a perceived need to finish tasks started by other team members.

On a higher level, adaptation to the resource situation did, as discussed in section 4.2, not happen before sprint 5 was completed. The changes implemented were effective and proved that the feedback loop worked. Nevertheless, this adaptation could ideally have happened earlier, as the team identified them in sprint 4, and the team had problems finishing with the desired functionality before the deadline. The somewhat late handling can be attributed to several practices. The overfull sprint backlogs probably obfuscated matters a bit, making the actual progress harder to measure. More significantly, the situation with many parallel projects also contributed, by demanding more evidence to implement effective measures because of the influence this would have on other projects.

5.2.6 Team Orientation

Team orientation is well facilitated by Scrum, which leaves much influence in the hands of the team. Shared planning procedures, and consensus are also important

Again, one of the main challenges for the team was limited hours available, due to parallel participation in other projects. The general impression was that although the team members was interested and tried to work as a team, these mostly organizational factors impeded the team effort. The team members had other priorities because of the quagmire, and the parallel projects. Team orientation suffered because of this, but it improved in the emergency phase. As an example, one developer suggested that he didn't really feel involved in the team before this phase.

5.3 Validity, Evaluation and Justification

Walsham [June 2006] suggests that the methodological approach is justified, preferably through the use of a pre-compiled criteria. One such is the seven principles of interpretive field research in "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems" [Klein, Heinz K. and Myers, Michael D., 1999].

5.3.1 Seven Principles

The Hermeneutic Circle

The hermeneutic circle requires iteration between considering the interdependent meaning of parts, and the whole these form.

This aspect has been important to understand this project. Parts such as the recurring large backlogs have been considered independently, and have further been connected to more underlying contextual causes. There will always be a possibility of not understanding the dependence of various parts, but generally this aspect has been given much consideration.

Contextualization

This principle requires reflection on the historical and social background of the research setting.

Historical and social elements were given much weight in this study. The actors experience and history with software engineering methods were important, especially their previous experience with Scrum and Agile methods. The social and organizational context was a essential piece of this case, and had a large impact on the analysis. While the explanation of the context could have been been a bigger part of this thesis, the most important aspects are covered sufficiently.

Interaction Between the Researchers and the Subjects

The researchers role in a study and interaction with the research subjects is critical to the understanding of the case. This principle requires a critical reflection on this aspect.

This study featured aspects of action-research and had an expressed focus on Scrum and Agile methods. This could very well have influenced the data gathered and statements given from the research subjects. They knew the interest was on Scrum and that their implementation of it was in part guided by the researchers. This could have lessened their rigour in the implementation and use, thinking that the researchers would let them know when something was wrong giving a sense of comfort. Some remarks from the findings support this suspicion.

On the other hand, the duration of the study and the spectrum of roles employed should have prevented this effect. Also, the interaction between researchers and the subjects is a recognized part of the study, as it is part of a action research project. While the findings might have been influenced, they should not be considered invalidated on behalf of this.

Abstraction and Generalization

Through the application of the first two principles, this principle requires that the data is interpreted through theoretical general concepts.

The two detailed research questions are contributions to generalization. RQ1 compares the findings with the theoretical Scrum principles. RQ2 uses the framework in Salas et al. [2005] to generalize the teamwork in this study, in order to add understanding on how a team functions in a Scrum project.

Dialogical Reasoning

A sensitivity to possible contradiction between theoretical preconceptions and the actual findings is the requirement for the principle of Dialogical reasoning.

The researchers preconceptions is recognized as a part of the study. While Scrum is the studied aspect, the longevity of the study, frequent discussions with co-researchers and feedback sessions with the subjects made reasoning on the findings versus existing theory a natural component of the process. Through this, theoretical preconptions are difficult to maintain.

Multiple Interpretations

Using multiple sources, is in this principle important to reveal possible differences in interpretations.

This is one aspect where sources could have been employed more effectively. While the interviews gave good data from the different actors in the project, as well as conversations underway, these differing narratives is quite possibly not used to the full extent in the resulting discussion. Some of the reasons for this can however be attributed to confidentiality reasons.

Suspicion

Sensitivity to "biases" and systematic "distortions" in collected narratives is the requirement for this principle.

There is a real possibility of this emerging in a study such as this, especially because the subjects knew they were studied and that it was possible that management would read the resulting report. Using several different data sources, and supplementing these with targeted interviews, as well as the long duration of the study should however have negated such distortions. It should be noted that the author felt that the subjects were up-front about their view through the entire study.

Chapter 6

The Final Table

Agile methods has in recent years gained a large following in industry. Research has been focused on XP, with few studies dedicated to Scrum.

This study has followed a software project employing Scrum in 8 sprints over a timespan of 10 months. The conclusions are based on data from several sources.

This chapter presents the final conclusions from the study, the generalized impact to research and practice, and proposes further work in the company and in academia.

6.1 Impact of Scrum in the project

The main impact of Scrum is that both developers and company is of the opinion that it is a useful contribution, similar to conclusions from general studies of Agile methods (see chapter 2).

Developers, both in the project studied and in other projects, were pleased with using Agile methods, and was clearly happy with prospect of continued use.

Also the company was positive, and initiated two other projects using Scrum during the study, which is considered as an indication that Scrum had a positive influence on process. While not all went as hoped in the project studied, it was considered as an improvement from previous projects.

Generally, teamwork functioned quite well, but also had some issues. The following sections elaborates on the specific conclusions which Scrum had on the project, divided by the research questions.

6.1.1 Implementation

> RQ 1: How is Scrum implemented in a project compared to the book? What are the benefits and disadvantages?

The study revealed a project which employed Scrum through the entire process. Despite limited initial training, most of the practices described in theory were implemented from the start with adjustments done to some of the issues discovered.

Good Project Visibility There was through the entire project high visibility into the process. The team managed to adjust their efforts according to this. While the adjustment could have been done earlier, the required adjustments were mostly external to the project, which made them harder to put in motion.

Low Opportunity + Little Need = Low Commitment There were several themes which indicated low sprint commitment; consistently large backlogs and no use of the burndown charts were two of the more visible. The low level of commitment was because of two connected reasons:

The organization had a high level of distraction, with developers involved in several projects at once, giving little opportunity to commit to sprints.

At the same time, the performance of sprints were measured partly by hours used, and increments were generally not put in use when finished. There was low urgency to finish sprints, because it was not expressed as important compared with the final deadline.

Questionable Predictability The project was set to deliver significantly behind schedule. Increments were delivered after each sprint, but the majority was clustered towards the end. This was mainly because of added resources, but can also be partly attributed to low commitment in the early sprints.

In contrast with other studies, the predictability was in this project relatively low. Predicting the final delivery date was difficult when the sprint commitment was low, combined with the practice of delivering almost potentially shippable code. These two practices were more similar to waterfall-planning of a project than the Agile way of thinking. If these had been eliminated earlier, the predictability of the project would probably have improved, with a better possibility to avoiding the slipped schedule.

6.1.2 Teamwork

RQ 2: How does Scrum support teamwork?

Regarding the teamwork in the project we can also draw some specific conclusions.

Communication! Developers were empowered. They specified, designed and developed the system from the start in cooperation with the management. As a whole, communication through the frequent meetings, coupled with initial work on GUI-sketches, ensured that the developers understood their task well. There was relatively little specialization among the developers. A shared mental model and closed loop communication was strong points in the entire project well supported by Scrum practices.

Adaptable Good communication had effects on mutual performance monitoring and back-up behaviour. As discussed in section 5.2 adaptability was high through the entire project, which had a significant positive impact. This is perfectly in sync with Scrum, and Agile theory, which holds adaptability as one of the main benefits.

Leadership Is Challenging In the project, the team orientation was relatively low, while some aspects of the leadership task could have been handled better. These two factors negated the effective mutual performance monitoring and backup behaviour.

It is quite clear that leadership is a challenge when using Scrum because of the distributed responsibilities between the different roles (see section 2.5.3). With the wide responsibility of ensuring the success of Scrum (see section 2.5.1), the Scrum master has several important but not necessarily well defined tasks.

Ensuring that the team is independent and works together were very successfully performed. On the other hand, the Scrum master has a role in creating sprints of the correct size, as well as making sure that increments are the main progress indicator. None of these tasks were performed optimally.

An additional problematic task was team protection, which was difficult due to contextual factors. Put together, the Scrum master focused more on ensuring communication and keeping a smooth process, while focusing less on maintaining and ensuring steady performance.

6.2 Implications For Research And Practice

As shown in chapter 3, it is possible to generalize on single-case studies. This section lays out the how this study has implications for the state of research, as well as the contribution to application of Agile methods in practice.

Research As mentioned in previous chapters, reviews of the state of research in Agile methods found that there have been limited scientific research on the use of Scrum. Abrahamsson et al. addressed an urgent need for empirical studies on the possibilities of Agile methods, and specifically on the adoption of practitioners. There exist in-depth studies on XP-projects, but none with Scrum as focus. By this, the main contribution of this thesis is the rich description of the use of Scrum in a real-world project.

The methods and data sources have been found appropriate for such a study, and are considered as well suited for this kind of research. The use of a framework for comparison and analysis was a valuable addition.

Practice Further, there are contributions relating to the more specific research questions:

One view expressed in chapter 2 was that Agile methods are easy to adopt and work well. This view is reinforced by this study, which suggests that Scrum is light enough to be used more or less as described, yet concise enough give valuable benefits. Contrary to XP, the findings suggest that Scrum works best as a whole, advising against pick-and-choose adoption of the method, and suggesting that tailoring of the method should happen over time. Scrum will mature through use, but sticking to the play-book is probably a good idea in the beginning.

One important issue was identified, which might be important depending on the specific setting. If Scrum is used in order to increase predictability, the findings suggests that this is difficult to achieve without a commited team and well-defined done-criteria. Commitment is strongly influenced by the environment of the project, as well as how the sprint deliveries are valued. Dybå and Dingsøyr writes that ease of implementation of Agile methods is affected by how interwoven the development is with other functions of the orgnaization, a view which correlates with this conclusion.

A point on the different roles of Scrum was revealed by the analysis of teamwork. Good interpersonal skills were found to be an important characteristic of successfull XP teams [Dybå and Dingsøyr, 2008]. This is also the case when using Scrum, but is especially true for the Scrum master.

This role complements the other roles, and has influence in almost all of the "Big Five of Teamwork". There is much responsibility with little official authority, and the often used comparison with a coach is found to be a surprisingly accurate methaphor. Education is an important, but interpersonal skills are vital in order to facilitate communication, while at the same time ensuring progress.

6.3 The Next Matches

With regard to the company, they are well underway to becoming a very agile organization. They do however have some challenges; There is a need for more continous quality improvement in the projects, which can be addressed through adoption of selected XP-practices. The biggest challenge is however in making sure that developers can focus on only one project at the time, without frequent interruptions from support and other parts of the organization. This was by far the biggest problem experienced in the study.

The author is reinforced in the initially positive view of Agile methods. It is however clear that while they are easier to implement than traditional methods, making them function optimally is difficult and requires perfection over time. Dicipline and professionalism are key.

Concerning further research, studies on teamwork in mature Scrum teams would be interesting. Another is to establish other major organizational obstacles, and the most common success factors relevant to Scrum adoption. Recognizing that Agile methods are people-centric, establishing these could help new teams to avoid common pit-falls. One thing which could be especially useful in this sense, is to establish a more formal evaluation framework, similar to Williams, L. et al. [2004], to facilitate comparison and to establish the real "*home grounds*" of Agile methods.

Bibliography

The engineers' council for professional development. *Science*, 94(2446):456, nov 1941. ISSN 0036-8075. URL `http://links.jstor.org/sici?sici=0036-8075%2819411114%293%3A94%3A2446%3C456%3ATECFPD%3E2.0.CO%3B2-F`.

P. Abrahamsson, O. Salo, J. Warsta, and J. Ronkainen. Agile software development methods: Review and analysis. *VTT Publications*, (478), 2002. ISSN 1235-0621.

R. Atkinson. Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management*, 17:337–342(6), 1999. doi: doi:10.1016/S0263-7863(98)00069-6.

D. E. Avison and G. Fitzgerald. Where now for development methodologies? *Commun. ACM*, 46(1):78–82, 2003. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/602421.602423.

K. Beck. Embracing change with extreme programming. *Computer*, 32(10): 70–77, Oct 1999. ISSN 0018-9162. doi: 10.1109/2.796139.

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. http://www.agilemanifesto.org/, 2001.

M. Beedle, M. Devos, Y. Sharon, K. Schwaber, and J. Sutherland. SCRUM: A pattern language for hyperproductive software development. In N. Harrison, B. Foote, and H. Rohnert, editors, *Pattern Languages of Program Design 4*, pages 637–652. Addison Wesley, 2000.

B. Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/12944.12948.

B. Boehm. Get ready for agile methods, with care. *Computer*, 35(1):64–69, 2002. doi: 10.1109/2.976920.

B. Boehm, D. Port, and A. W. Brown. Balancing plan-driven and agile methods in software engineering project courses. *Computer Science Education*, 12(3):187–195, sept 2002. ISSN 1744-5175.

F. P. J. Brooks. No silver bullet: essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987. ISSN 0018-9162.

A. Bryant. It's engineering jim ... but not as we know it: software engineering - solution to the software crisis, or part of the problem? In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 78–87, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-206-9. doi: http://doi.acm.org/10.1145/337180.337191.

G. Cloke. Get your agile freak on! agile adoption at yahoo! music. *agile*, 0: 240–248, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/AGILE.2007.30.

A. Cockburn. *Agile software development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0-201-69969-9.

D. Cohen, M. Lindvall, and P. Costa. An introduction to agile methods. *ADVANCES IN COMPUTERS, VOL 62*, 62:1 – 66, 2004. ISSN 0065-2458.

T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 2008. doi: 10.1016/j.infsof.2008.01.006.

B. Fitzgerald, G. Hartnett, and K. Conboy. Customising agile methods to software practices at intel shannon. *EUROPEAN JOURNAL OF INFORMATION SYSTEMS*, 15(2):200 – 213, 2006. ISSN 0960-085X.

R. D. Galliers and F. F. Land. Viewpoint: choosing appropriate information systems research methodologies. *Commun. ACM*, 30(11):901–902, 1987. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/32206.315753.

N. Haugen. An empirical study of using planning poker for user story estimation. *Agile Conference, 2006*, pages 9 pp.–, 2006. doi: 10.1109/AGILE.2006.16.

I. T. Hawryszkiewycz. *Introduction to systems analysis and design*. Pearson Education Australia, 5 edition, 2001. ISBN 1-7400-9280-5.

A. Highsmith, J.; Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120–127, Sep 2001. ISSN 0018-9162. doi: 10.1109/2.947100.

IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, 1990. URL `http://ieeexplore.ieee.org/servlet/opac?punumber=2238`.

Klein, Heinz K. and Myers, Michael D. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 23(1):67–93, mar 1999. ISSN 0276-7783. URL `http://links.jstor.org/sici?sici=0276-7783%28199903%2923%3A1%3C67%3AASOPFC%3E2.0.CO%3B2-Q`.

C. Larman and V. R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, 2003. ISSN 0018-9162.

A. B. B. Lervåg. A case study of a norwegian scrum project. Master's thesis, NTNU, 2006.

C. Mann and F. Maurer. A case study on the impact of scrum on overtime and customer satisfaction. *adc*, 0:70–79, 2005. doi: http://doi.ieeecomputersociety.org/10.1109/ADC.2005.1.

Miles, Matthew B. and Huberman, Michael A. *Qualitative Data Analysis: An expanded Sourcebook*. SAGE Publications, 2 edition, 1994. ISBN 0-8039-4653-8.

N. B. Moe and T. Dingsøyr. Scrum and team effectiveness: Theory and practice. 2008. Submitted to XP 2008.

M. Myers. Investigating information systems with ethnographic research. *Commun. AIS*, page 1.

M. D. Myers and L. W. Young. Hidden agendas, power and managerial assumptions in information systems development: An ethnographic study. *Information Technology & People*, 10:224–240(17), 1997.

S. Nerur, R. Mahapatra, and G. Mangalaraj. Challenges of migrating to agile methodologies. *Commun. ACM*, 48(5):72–78, 2005. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1060710.1060712.

H. Robinson, J. Segal, and H. Sharp. Ethnographically-informed empirical studies of software practice. *Inf. Softw. Technol.*, 49(6):540–551, 2007. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2007.02.007.

K. H. Rolland and J. Herstad. The 'critical case' in information systems research. URL `http://citeseer.ist.psu.edu/416909.html`.

W. W. Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. ISBN 0-89791-216-0.

E. Salas, D. E. Sims, and C. S. Burke. Is there a "Big Five" in Teamwork? *Small Group Research*, 36(5):555–599, 2005. doi: 10.1177/1046496405277134. URL `http://sgr.sagepub.com/cgi/content/abstract/36/5/555`.

O. Salo. *Enabling Software Process Improvement in Agile Software Development Teams and Organisations*. PhD thesis, University of Oulu, 2007.

K. Schwaber. What is scrum? Published by the Scrum Alliance. URL `http://www.scrumalliance.org/system/resource/file/275/whatIsScrum.pdf`.

K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001. ISBN 0130676349.

H. Sharp and H. Robinson. An ethnographic study of xp practice. *EMPIRICAL SOFTWARE ENGINEERING*, 9(4):353 – 375, 2004. ISSN 1382-3256.

I. Sommerville. *Software Engineering*. Addison Wesley, 1995. ISBN 0-201-42765-6.

I. Sommerville. Software process models. *ACM Comput. Surv.*, 28(1):269–271, 1996. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/234313.234420.

H. Takeuchi and I. Nonaka. The new new product development game. *Harvard Business Review*, 64(1):137–146, 1986.

S. E. Tøsse. Kodekultur – programmeringens iscenesettelser. Master's thesis, NTNU, 2007.

G. Walsham. Knowledge management: The benefits and limitations of computer systems. *European Management Journal*, 19(6):599–608, 2001.

G. Walsham. Interpretive case studies in is research: nature and method. *European journal of information systems*, 4(2):74–81, 1995.

G. Walsham. Doing interpretive research. *European Journal of Information Systems*, 15:320–330(11), June 2006. doi: doi:10.1057/palgrave.ejis.3000589.

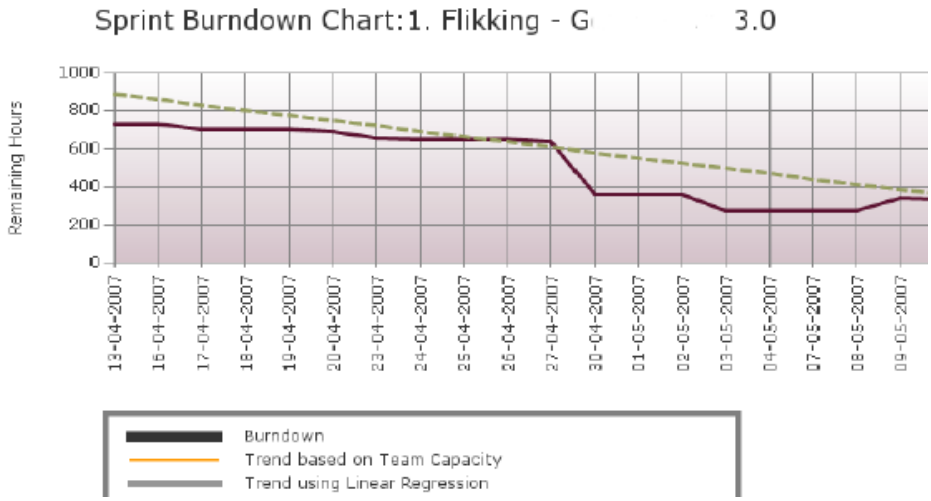
Williams, L., Layman L., and Krebs W. Extreme programming evaluation framework for object-oriented languages – version 1.4. Technical report, North Carolina State University Department of . Computer Science, 2004.

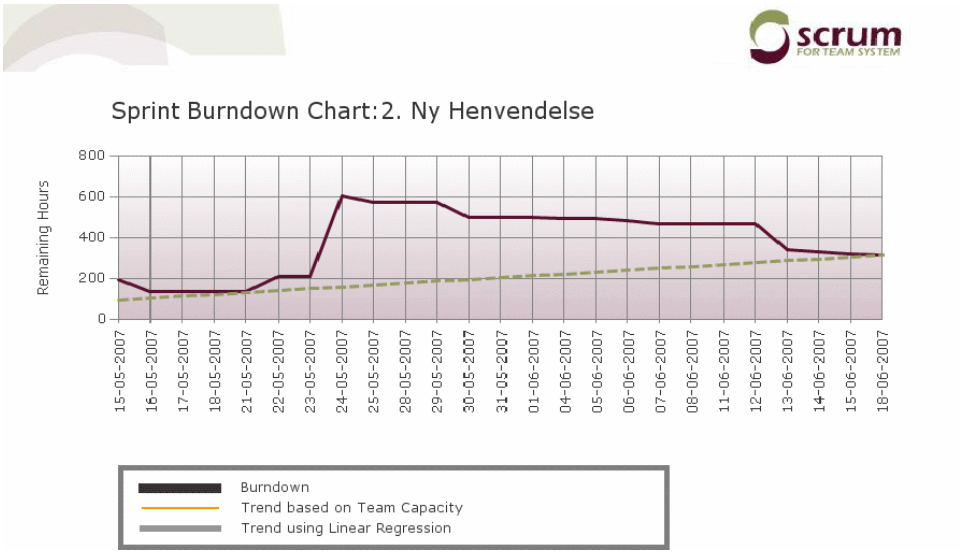
WordNet. Wordnet - princeton university cognitive science laboratory. www. URL `http://wordnet.princeton.edu/perl/webwn`.

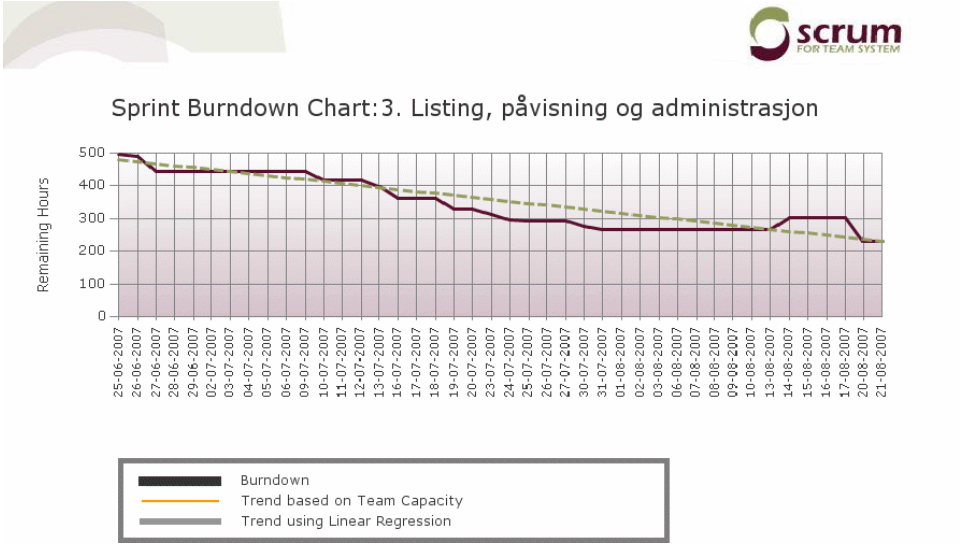
Appendix A

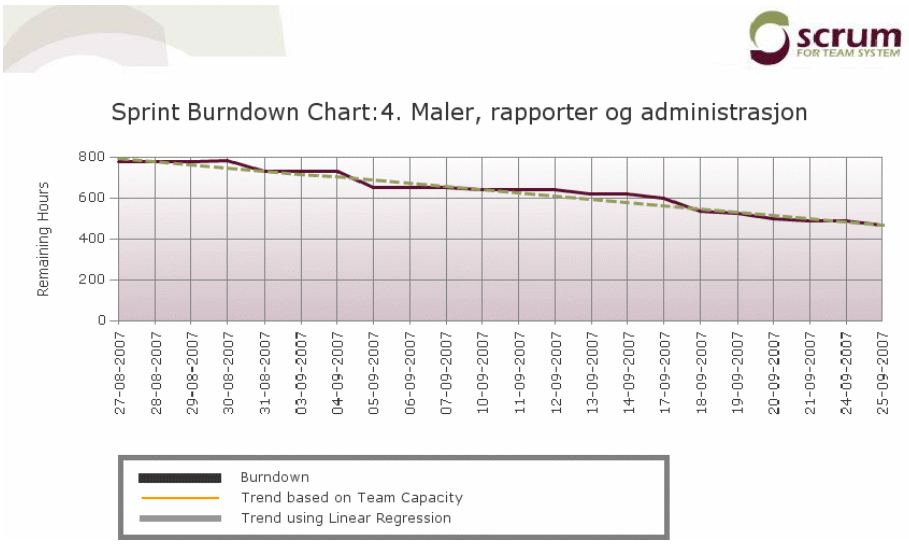
Quantitative Data

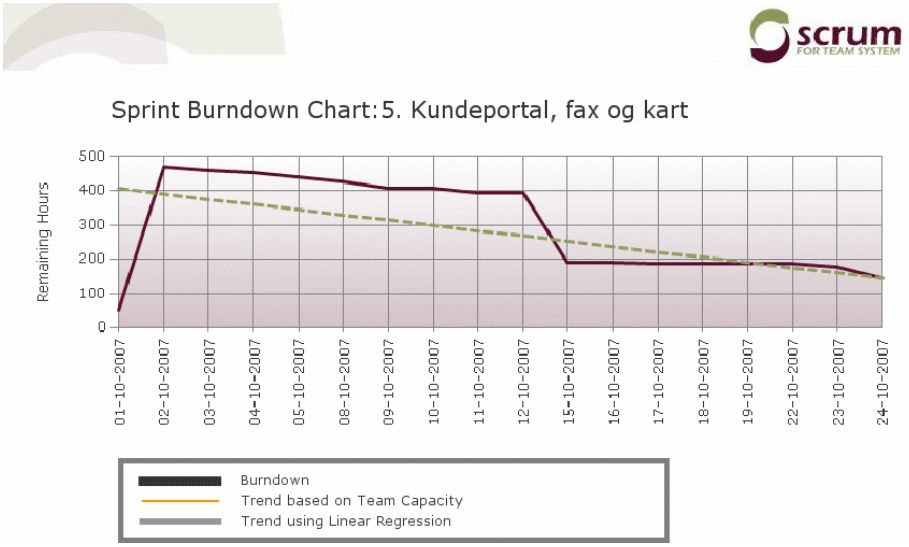
A.1 Burndown Charts

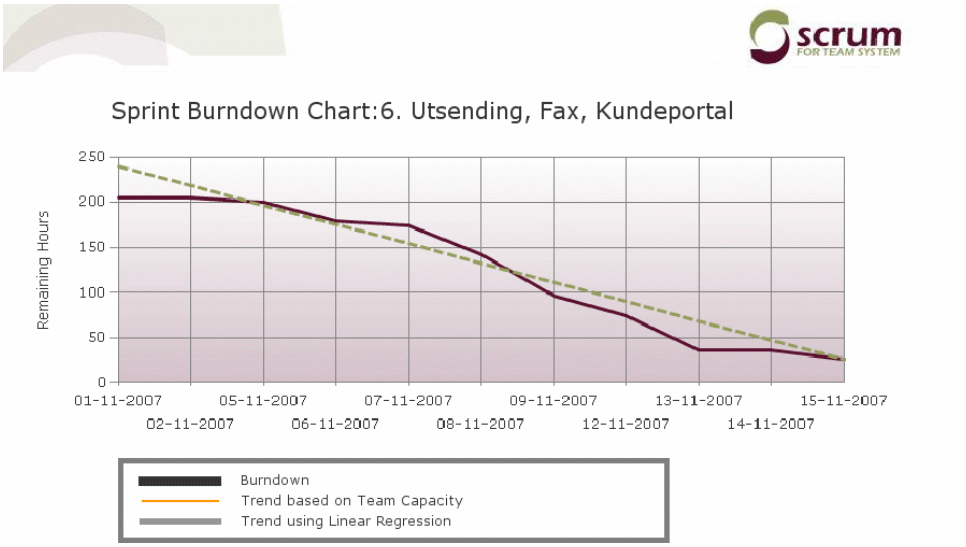


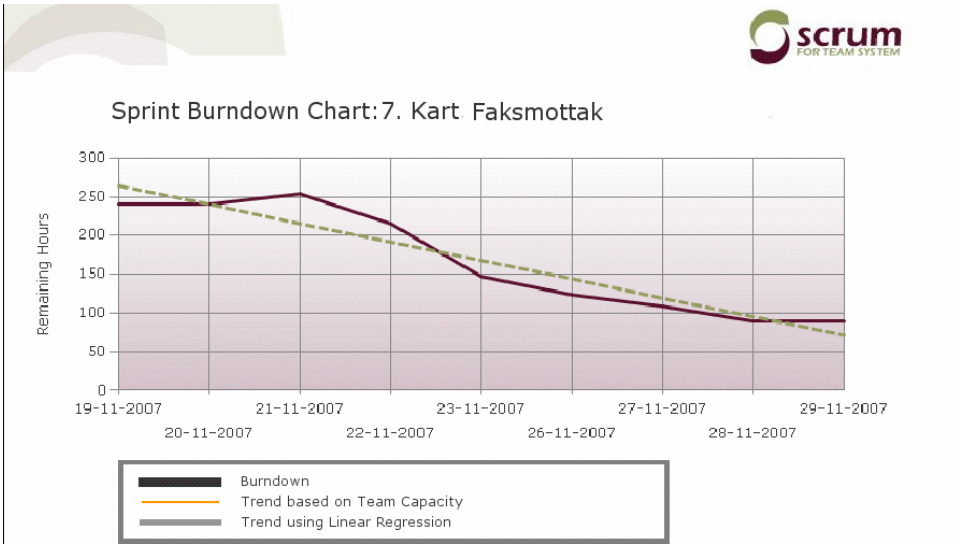

scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:2. Ny Henvendelse
Remaining Hours
800
600
400
200
0
15-05-2007
16-05-2007
17-05-2007
18-05-2007
21-05-2007
22-05-2007
23-05-2007
24-05-2007
25-05-2007
28-05-2007
29-05-2007
30-05-2007
31-05-2007
01-06-2007
04-06-2007
05-06-2007
06-06-2007
07-06-2007
08-06-2007
11-06-2007
12-06-2007
13-06-2007
14-06-2007
15-06-2007
18-06-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression

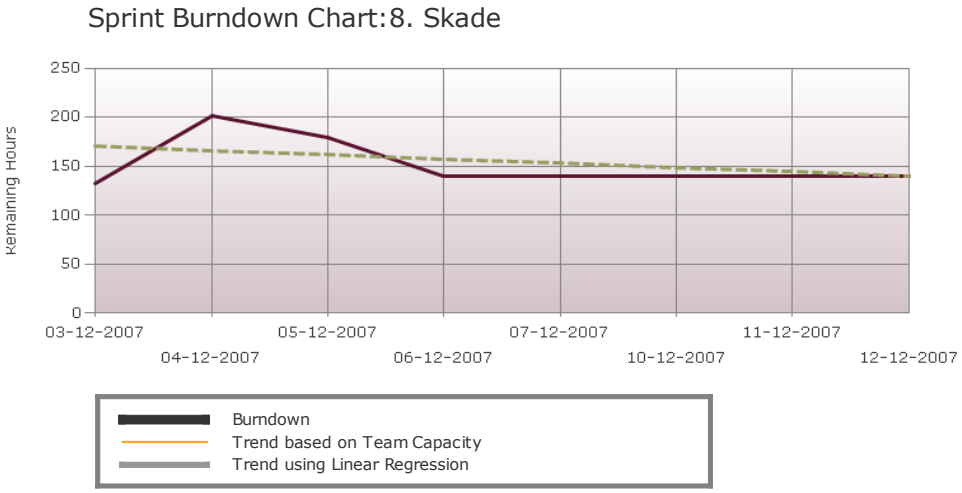

scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:3. Listing, påvisning og administrasjon
Remaining Hours
500
400
300
200
100
0
25-06-2007
26-06-2007
27-06-2007
28-06-2007
29-06-2007
02-07-2007
03-07-2007
04-07-2007
05-07-2007
06-07-2007
09-07-2007
10-07-2007
11-07-2007
12-07-2007
13-07-2007
16-07-2007
17-07-2007
18-07-2007
19-07-2007
20-07-2007
23-07-2007
24-07-2007
25-07-2007
26-07-2007
27-07-2007
30-07-2007
31-07-2007
01-08-2007
02-08-2007
03-08-2007
06-08-2007
07-08-2007
08-08-2007
09-08-2007
10-08-2007
13-08-2007
14-08-2007
15-08-2007
16-08-2007
17-08-2007
20-08-2007
21-08-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression


scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:4. Maler, rapporter og administrasjon
Remaining Hours
800
600
400
200
0
27-08-2007
28-08-2007
29-08-2007
30-08-2007
31-08-2007
03-09-2007
04-09-2007
05-09-2007
06-09-2007
07-09-2007
10-09-2007
11-09-2007
12-09-2007
13-09-2007
14-09-2007
17-09-2007
18-09-2007
19-09-2007
20-09-2007
21-09-2007
24-09-2007
25-09-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression


scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:5. Kundeportal, fax og kart
Remaining Hours
500
400
300
200
100
0
01-10-2007
02-10-2007
03-10-2007
04-10-2007
05-10-2007
08-10-2007
09-10-2007
10-10-2007
11-10-2007
12-10-2007
15-10-2007
16-10-2007
17-10-2007
18-10-2007
19-10-2007
22-10-2007
23-10-2007
24-10-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression


scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:6. Utsending, Fax, Kundeportal
Remaining Hours
250
200
150
100
50
0
01-11-2007
02-11-2007
05-11-2007
06-11-2007
07-11-2007
08-11-2007
09-11-2007
12-11-2007
13-11-2007
14-11-2007
15-11-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression


scrum
FOR TEAM SYSTEM
Sprint Burndown Chart:7. Kart. Faksmottak
Remaining Hours
300
250
200
150
100
50
0
19-11-2007
20-11-2007
21-11-2007
22-11-2007
23-11-2007
26-11-2007
27-11-2007
28-11-2007
29-11-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression


Sprint Burndown Chart:8. Skade
Remaining Hours
250
200
150
100
50
0
03-12-2007
04-12-2007
05-12-2007
06-12-2007
07-12-2007
10-12-2007
11-12-2007
12-12-2007
Burndown
Trend based on Team Capacity
Trend using Linear Regression

A.2 The project in numbers

	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5
Time period	11.4–11.5	15.5–18.6	25.6–21.8	27.8–25.9	1.10–25.10
Work days	22	24	43	22	20
Lines (x1000)	82	34	81	44	15
Churn (x1000)	102	56	147	54	18
Prod. backlog items	4	7	33	29	34
Sprint backlog tasks	17	14	38	37	54
Orphan tasks	12	7	5	6	8
Initial sprint tasks	40	43	66	82	56
Tasks completed	8	13	18	22	15
Hrs (est. ex def)	340	290	267	355	297
Hrs (est. ex orph)	55	82	158	348	270
Rem hrs. (est.)	42	16	0	6	34
Initial hours (est.)	762	720	624	793	486
Hours (est.) remaining	479	323	417	505	327
Hours available	300	350	327	400	350
Hours used (actual)	182	245	372	375	260
Completed hours (est.)	298	274	267	349	263
Brkdown factor	1,25	1,00	1,00	1,07	1,35
Resources	61 %	70 %	114 %	94 %	74 %
Avg. hrs per day	8,27	10,21	8,65	17,05	13,00
Estimation acc.	61 %	89 %	139 %	107 %	99 %
Avg. item size (est. hrs)	85,00	41,43	8,09	12,24	8,74
Avg, task size (est. hrs)	20,00	20,71	7,03	9,59	5,50

	Sprint 6	Sprint 7	Sprint 8
Time period	1.11–15.11	19.11–29.11	3.12–12.12
Work days	11	9	8
Lines (x1000)	21		
Churn (x1000)	28		
Prod. backlog items	12	35	38
Sprint backlog tasks	15	45	47
Orphan tasks	0	7	7
Initial sprint tasks	12	46	28
Tasks completed	6	33	9
Hrs (est. ex def)	321	195	269
Hrs (est. ex orph)	281	183	228
Rem hrs. (est.)	8	0	100
Initial hours (est.)	207	246	242
Hours (est.) remaining	36	114	148
Hours available	254	254	
Hours used (actual)	278	253	
Completed hours (est.)	313	195	169
Brkdown factor	1,25	1,09	1,05
Resources	109,4%	99,6%	
Avg. hrs per day	25,3	28,1	0,0
Estimation acc.	88,8%	129,7%	0,0%
Avg. item size (est. hrs)	26,8	5,6	7,1
Avg, task size (est. hrs)	21,4	4,3	5,7

	Avg. S1-S5	Avg. S6-S8	Average	Total
Time period				
Work days	26,2	9,3	19,9	159
Lines (x1000)	51,2			277
Churn (x1000)	75,4			405
Prod. backlog items	21,4	23,5	24,0	192
Sprint backlog tasks	32	30	33,4	267
Orphan tasks	7,6	3,5	6,5	52
Initial sprint tasks	57,4	29	46,6	373
Tasks completed	15,2	19,5	15,5	124
Hrs (est. ex def)	309,8	258	291,8	2334
Hrs (est. ex orph)	182,6	232	197	
Rem hrs. (est.)	19,6	4	15	
Initial hours (est.)	677	226,5	510	4080
Hours (est.) remaining			314	
Hours available	345,4	254	279	2235
Hours used (actual)	286,8	265,5	246	1965
Completed hours (est.)	290,2	225,7	266	2128
Brkdown factor	1,13	1,13	1,13	
Resources	82,5%	69,7%	88,8%	
Avg. hrs per day	11,4	17,8	15,8	
Estimation acc.	98,8%	104,5 %	89,34 %	
Avg. item size (est. hrs)	14,5	11,0	24,4	194,9
Avg, task size (est. hrs)	12,6	12,8	11,8	94,3

Appendix B

Research templates

Write-up: Week X

Encounters

Date	Researcher	Type of contact

Summary

Events (project)

Process

Key observations

Quotes

Intervjuguide

Generelt

1. Hvordan har dette prosjektet vært sammenligna med andre prosjekter?

2. Hvor viktig har dette prosjektet vært?

3. Hvordan synes du kvaliteten er på det som har blitt utviklet?

4. Hva har fungert bra i prosjektet?

5. Hva har ikke fungert så bra i prosjektet?

Scrum: Møter

1. Hva synes du om de forskjellige møtene som er en del av Scrum metodikken?
   - Sjekk at man snakker litt om de daglige møtene?

Scrum: Planlegging og Estimering

1. Hvordan synes du estimeringsprosessen har blir gjennomført?

2. Hvordan synes du prosjektet er planlagt. Fungerer planlegging ved bruk av backloggene?

3. Hvor viktig har det vært å fullføre sprint backlog?
   - Sjekk: henger dette sammen med størrelse på sprint backlog?

4. Sjekk: Har dette forandret seg etter sprekken?

5. Synes du Burndown er nyttig?

Verktøybruk

1. Hvordan har Team System fungert

2. Hvordan har veggen og kartotekkortene fungert?

3. Har disse to fungert bra sammen?

4. Hva ville du valgt om bare ett skulle brukes?

5. Hvor viktige har Scrum-verktøyene vært for prosjektet?
   - Verktøy = Team system + Visual Studio samt indexkort+burndown.

Teamarbeid

1. Hvordan har oppgaver blitt oppdelt og tildelt?

2. Påvirker hengemyra denne oppdelinga?

3. Kan du fortsette på arbeidet om et av teammedlemmene blir truffet av Gråkalltrikken?
4. Er det noe annet du som er viktig men som vi ikke har snakket om?