NTNU
Innovation and Creativity

A Case Study of a Norwegian Scrum Project

Alf Børge Bjørdal Lervåg

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Eric Monteiro, IDI

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

Problem Description

Do a case study of an agile project in Norway.

Assignment given: 20. January 2006
Supervisor: Eric Monteiro, IDI

Abstract

In this paper I present a case study of a Norwegian development project where the development team adopted practices from Scrum in the middle of the development effort. My study shows that the developers were happy with this new development method, and among other things thought it gave them a better focus and structure for their work. Since the team only adopted practices from the Scrum method, I look at the differences between their method and Scrum and suggest a few improvements to their method based on my knowledge of Scrum and development methods in general.

Preface

I wanted to learn more about agile software development, and after a few talks with my teaching supervisor (Monteiro), we agreed to search for a live project I could use for a case study. Monteiro got me in contact with Torgeir Dingsøyr from *SINTEF Department of Software Engineering, Safety and Security (SINTEF)* who, despite his paternity leave, agreed to contact some of the companies he was involved with and ask if they could help me out.

Since proximity would make the study easier, we tried to find companies that were located in Trondheim first, but in the end we had to broaden our search to include Oslo. Both Dingsøyr and I were searching for companies during this time, but neither of us found any good prospects.

Just when I was about to give up and study an open source software project instead, Dingsøyr contacted me and said he had found a project for me. After this things moved very fast. Two researchers at SINTEF combined the start of my study with the start of one of their own projects and booked a meeting with Leader 1 and Senior 2 from Company 2. This meeting was primarily for the researchers from SINTEF but it gave me a convenient entrance to Company 2.

Acknowledgements

First of all I would like to thank Developer 1 for his quick and comprehensive replies to my questions during the last weeks before my deadline. Many thanks to my teaching supervisor, Eric Monteiro, for his advice and help with making this paper possible. Torgeir Dingsøyr, without whom I would not have had a project to study; Tore Dybå and Geir Kjetil Hanssen from SINTEF who let me piggyback on one of their own projects to give me a comfortable introduction to Company 1. I would like to thank Leader 1 who opened the doors for me, all the people I talked with at Company 1 and finally Customer 1 and the development team who treated me so well during the study.

I'm very grateful for the help from the people who read through the drafts of

my paper and gave me valueable feedback; Magni Onsøien, Eli Toftemo, Einar Ryeng, Mathiasm Lidal and Truls Tangstad. Thanks for pointing out my stupid errors without poking too much fun at me.

Contents

List of Figures

Chapter 1

Introduction

Do you know what agile software development is? Does it sound familiar? I suspect it does. Recently agile methods have become more and more popular. A search on google for "agile software development" gives 16 000 000 results. What is often considered the opposite of agile is sequential software development. Just for fun, I did a search for this on google and it gives 10 600 000 results. While this doesn't prove anything, it does indicate that agile software development has indeed become popular.

How about Scrum? No, I'm not talking about rugby[1], I'm talking about the software development method. Since google only gives 629 000 results when searching for "scrum software development" I guess you might not have heard of it. Neither had I before I started this project.

Anyway, in Chapter 2 I give an introduction to these concepts so don't worry if this was completely new to you.

When we're all on the same page regarding development methods, I tell the story of this little project and the people involved. This is covered in Chapter 3. Here I look at how a small team of developers in Norway adopted parts of the Scrum method during their project. By choosing to adopt only what they felt was necessary and at their own pace, they are now exploiting some of the benefits of this

[1] In case you're not familiar with rugby, Scrum is the name of how a game begins

agile development method.

In Chapter 4 I discuss some the research methods I have used to conduct this study.

Finally, in Chapter 5, I try to analyze the results in light of development theory and the books and articles I have read concerning how to best develop software.

Chapter 2

Development Methods

A development method is commonly referred to as a collection of ideas and practices for planning, designing and developing IT-systems. The word methodology and model is also sometimes used to describe the same thing, but since these different terms cause more confusion than clarification I will avoid the term methodology completely, and limit my use of the word model to where it is part of the name of the method I write about.

McConnell says that any approach to programming constitutes a method no matter how unconscious or primitive the approach is [McC04, p657]. He adds that the point of most methods is to reduce communication problems (see Section 2.1.1).

I start this chapter with a few definitions and explanations to avoid misunderstandings. Then I cover the sequential development methods, the incremental development methods and finally the agile development methods. For each of these I give one or two examples. Finally I use an entire section on the Scrum method since this is the development method that was used in the project I studied.

2.1 Definitions and Explanations

In this chapter and the chapters that follows I will use a few terms that in some cases mean different things to different people. See for instance the first paragraph in this chapter for an example. To avoid misunderstandings I try to define the words and terms I use that I believe might cause problems.

2.1.1 Formalism

As the size of the development team increases, McConnell argues that the number of communication paths increases multiplicatively, proportionally to the square of the number of people [McC04, p650]. As the number of communication paths increases, so does the amount of time spent communicating and the risk of communication mistakes increases. His conclusion is that larger-sized projects need a way to streamline communication or limit it in a sensible way.

A typical approach for streamlining communication is to formalize the communication in documents. Different development methods have different levels of formalism, as illustrated in Figure 2.1.

Formalism is also dependant on the criticality of the project, as argued by Cockburn [Coc02, p162]. If the consequence of a failure in the program will lead to injuries or death then the required level of formalism is higher than if the result is a few hours of lost work.

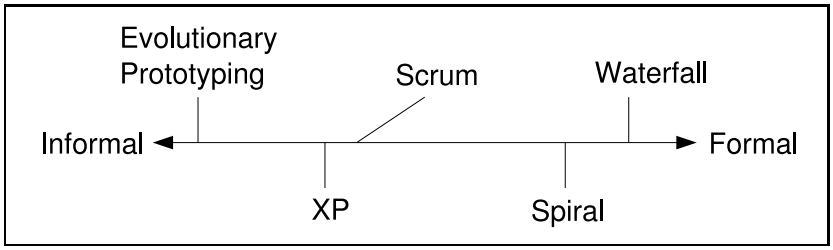

Figure 2.1: Development methods have different levels of formalism. The Waterfall model (see Section 2.2.1) is very formal, while Evolutionary Prototyping (see Section 2.3.2) is very informal.

2.1.2 Iterative and Incremental

In the article *Iterative and Incremental Development: A Brief History* [LB03], Larman and Basili makes a very good summary of the history of Iterative and Incremental Development. Here they write that while some prefer to reserve the phrase *iterative development* merely for rework, it usually also implies evolutionary advancement.

Despite their definition I have decided to use the phrase *iterative development* for rework, and the phrase *incremental development* for evolutionary advancement since this makes it easier for me to separate between the concepts.

2.2 Sequential Development Methods

The Sequential Development methods are, as the name implies, methods that go through a set of phases sequentially until the software is complete and delivered. These methods usually include, but are not limited to

- Requirements Specification
- Design
- Code
- Test
- Release

They have been very popular in some fields since they make it easier to write concrete contracts where the developers promise to deliver the product specified in the requirements specifications within a certain deadline. This makes it easier to make a budget for the project and the corporation feels safe because they have a binding contract promising a delivery.

Most sequential methods assume that you can predict the requirements and design needs early in the development project, and that these do not change during the project life time. If this assumption is true, which is probably the case for some

but far from all projects, then it is obviously best to collect the requirements and plan a design that fits these requirements as early as possible.

However, experience has showed that it is very difficult, if not impossible, to make perfect predictions. Usually, one encounters issues and learn new things that make it necessary to redo or change the plans during development. Also, with sequential methods it is very difficult to exploit what the developers learn during development since the cost of changing the requirements and design is too high to allow anything but critical changes.

There are several reports that show that sequential methods have a high risk of failure. According to Larman and Basili [LB03], the Standish Groups "CHAOS: Charting the Seas of Information Technology" report [Joh99] that looked at 23 000 projects to determine failure factors shows that the top reason for failure was associated with "waterfall practices" (sequential development). Another study of a sample of the Department of Defences (DoD) software spendings in 1995 by Stanley J. Jarzombek [Jar99] shows that 75% of the projects failed or were never used. Note that DoD projects were expected to use a sequential development method at that time period.

Leisham and Cook [LC02] makes a convincing argument of how the process of gathering requirements needs to be iterative, since the chance of getting it right on the first try is minimal. Often the customer has preconceived notions that he considers obvious and therefore doesn't mention on the first round, and what is said by the customer isn't always what the developer hears. The result is that the requirements are wrong or incomplete and the resulting product does not deliver what is needed.

Most sequential methods have a high level of formalism, usually in the form of separate documents for each phase of development. Because of this they are sometimes referred to as document driven or plan-driven methods.

2.2.1 The Waterfall model

Royce presented what has later become known as *The Waterfall model* in his paper "Managing the Development of Large Software Systems" [Roy70] in 1970. It divides the software development process into seven distinct phases that are sequentially dependant on each other, see Figure 2.2. Royce argues that there is a need for iterative interaction between the various phases and that these iterations should not be confined to the successive steps. This argument was based on Royces experience that predictions and planning often are wrong or incomplete and that when this is discovered at later phases there is a need to go back and fix the problem before moving on.

It should be noted that Royce wrote his article under the constraint of the government-contracting models, and that he himself was a supporter of incremental development (see Section 2.3) [LB03].

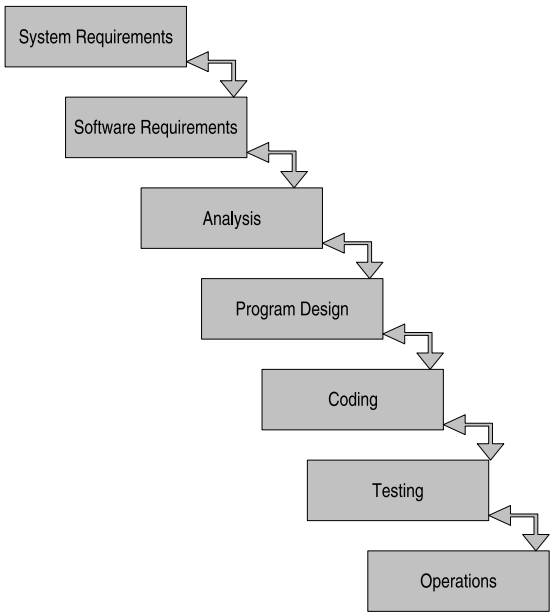

Figure 2.2: What is usually considered the Waterfall model. This is actually a simplified model presented by Royce [Roy70]. In the article Royce presents this simplification only to criticise it and present an improved model.

As mentioned earlier, much of the argumentation for using the Waterfall model

(and sequential methods in general) is based on Boehms Cost of Change Curve (See Figure 2.3) which is presented in his book “Software Engineering Economics” [Boe81]. He shows that the cost of changing requirements or fixing defects rises exponentially as the project nears completion. Boehm argues that since changes are much cheaper early in the project one should use extra time here and, with the help of proper planning, minimize the necessary changes later when they are too expensive.

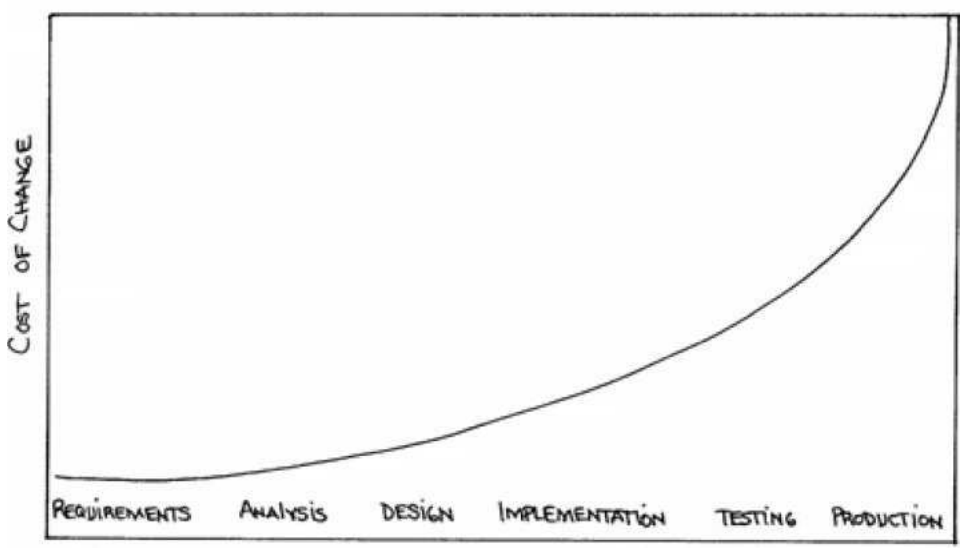

Figure 2.3: Boehms Cost of Change Curve, a graphical representation of how the cost of fixing defects rises exponentially the later it is done in the development project. (Figure taken from [Bec00, p21].)

2.3 Incremental Development

As mentioned above, most sequential methods contain some iterations or rework of previous steps based on knowledge acquired in later steps. Still, the plan is to deliver the complete product in one go, usually after years of development. When using *incremental development* on the other hand, the plan is to do several iterations where each iteration accomplishes something of value.

Hans van Vliet [vV00, 52] writes that if users are shown a working system at an

early stage and are given the opportunity to try it out, problems are detected at an early stage as well. Giving users a chance to influence and modify the design will help make the system features reflect the users real requirements and make the system easier to use.

Most modern development methods incorporate incremental development, but I will limit myself to presenting the classical *Spiral model* and a more drastic approach called Evolutionary Prototyping.

2.3.1 The Spiral model

Boehms Spiral model [Boe88] is the classical example of an incremental method. It builds on the Waterfall model, but instead of being document oriented it is risk oriented. In each increment you identify the sub-problem which constitutes the biggest risks, and then you resolve this.

The Spiral model adapts itself based on the risks involved and can be coupled with the various methods discussed in this chapter by focusing on different risks. [vV00, p62]. For instance, in a project where the user interface and performance requirements are considered low risk while the budget and schedule predictability and control are high risk, the Spiral model would result in something that looks like a sequential method. However, in a project where the user interface or user decision support requirements are high risk while the budget and schedule predictability and control are low risk, then the Spiral model will adapt and be more equivalent to the evolutionary methods (see below).

Figure 2.4 should make it obvious why the method is called the Spiral model. Note how the cost of the project increases as the project moves on. One of the important decisions that should be done in the risk analysis step is to decide whether the project should be terminated or not. This gives the stake holders a chance to cancel the project if the project leader can't convince them that it will be profitable.

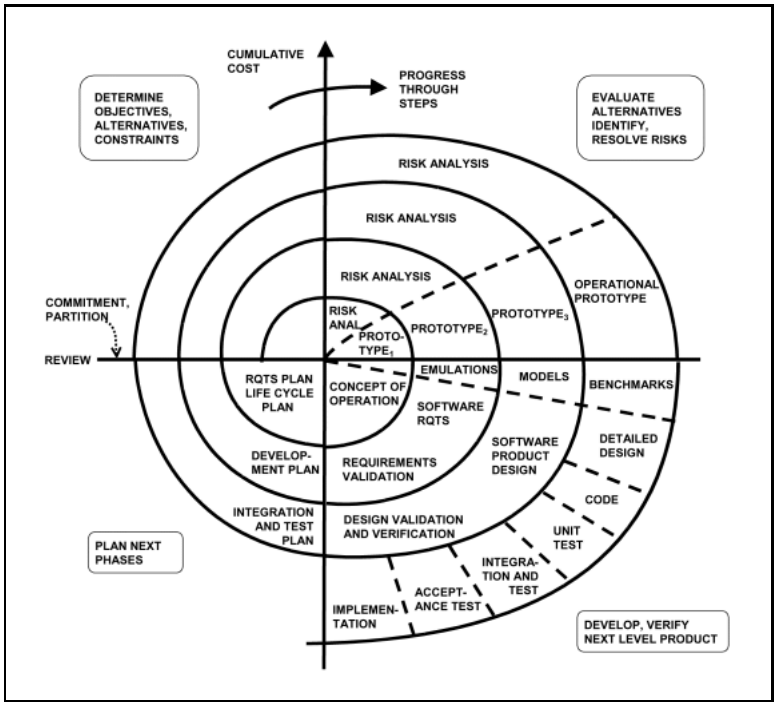

Figure 2.4: Boehms Spiral model of the software process

2.3.2 Evolutionary Prototyping

Using the term evolutionary in regard to development was introduced by Tom Gilb in his book “Software Metrics” [Gil76] in 1976. According to Larman and Basili [LB03] it is one of the first books to discuss the themes incremental design and evolutionary delivery properly.

Gilb writes that

> A complex system will be most successful if it is implemented in small steps and [...] each step has a clear measure of successful achievement as well as a “retreat” possibility to a previous successful step upon failure. You have the opportunity of receiving some feedback from the real world [...], and you can correct possible design errors [...].

The development team is supposed to work close together with the user and discover the requirements based on discussions and cooperation. This helps the user

feel more involved in the project and leads to a deeper understanding of what is possible, something that can lead to higher quality requirements [vV00, 55].

McCracken and Jackson describes this in their article “Life Cycle Concept Considered Harmful” [MJ82]. They recommend delivering a product for experimentation or actual use, based on the earliest and most tentative requirements of the customer. Development then proceeds in cooperation with the user as insight into the user’s own environment and needs is accumulated. The development proceeds with delivering a series of modifications to the first prototype which gradually evolves into the final product.

They point out that when using this method a formal specification might be unnecessary, and that the prototype itself can furnish this specification if there is a need to reimplement the product for some reason.

2.4 Agile Development Methods

The dictionary[1] defines agile as follows

> ag-ile (adj.)
>
> 1. Characterized by quickness, lightness, and ease of movement; nimble.
> 2. Mentally quick or alert: an agile mind.

Even though methods that can be classified as agile have existed since at least 1968 [LB03], the term agile software didn’t exist until 2001 when it was coined by the Agile Alliance [All01]. The founders of the Agile Alliance were all working on different software development methods that tries to handle the problem with unpredictable and unstable requirements. Naturally, they didn’t agree on a single method for developing software, but they did agree on the agile manifesto (see below) and to use the word agile to describe the similarities between their methods.

[1] I used `www.dictionary.com` [Online; accessed 17-December-2005]

The agile methods assumes that there will be unpredictable changes during the development and that it is better to focus on ways to adapt and handle these changes instead of trying to predict them. Kent Beck [Bec00] uses the term *embrace change* to describe this attitude.

While these methods can be considered a subset of the incremental methods, there are several things that make them stand out and supports the choice of gathering them under a new name. The agile manifesto [All01] by the founders of the Agile Alliance tries to define this:

> 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.

See Cockburn [Coc02, p216–218] for an explanation of the agile manifesto and its implication.

One of the things the agile methods have in common is the use of very short iterations. Where the Spiral model is usually used with iterations from 6 months to 2 years [Boe88], most agile methods advocate iterations shorter than 1 month [Coc02, p179]. The reason for this is that shorter iterations cause less damage (cost) if the result after the iteration is defective (misunderstandings or design flaws). It also makes it easier to accommodate customer collaboration since the customer can be involved and see the progress at the end of each iteration, and is even asked whether he would like to change the priorities or requirements for the next iteration.

Agile methods also aim for a formality level that is as low as possible. This is a result from the agile manifesto. The focus is to create working software, not

documentation. Reducing the amount of documentation (formalism) as much as possible frees time to make better software. Note that this doesn't mean that you shouldn't write documentation. Some documentation is always required and should be delivered, however documentation that isn't used shouldn't be written. [Coc02]

2.4.1 Extreme Programming (XP)

> XP is a lightweight [method] for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.
> – Kent Beck [Bec00, p.xv]

The core of XP is the so called enabling practices as described by Fowler [Fow01] and presented below. These practices could be said to flatten the change curve described above enough to defend a simple design for today instead of a complex design for tomorrow [Bec00]. Later argumentation by Cockburn [Coc00] claims that XP in fact doesn't flatten this curve but that the flattened curve could be seen as a representation of the Cost of Change in XP compared to sequential development. However, the practices increases the chances of catching problems as early as possible and fixing them as soon as they are caught [Coc00].

The practices of XP are not new or revolutionary as Beck writes in his article "Embracing Change With Extreme Programming" [Bec99]. What's new in XP is the insight that the practices work better together and that being disciplined in using these practices will make it possible to adapt to changing requirements.

Notice how for instance refactoring is made possible because of continuous integration and testing, how testing and simple design is enforced by pair programming and so forth.

Beck enumerates 13 practices in his article [Bec99] and 12 practices in the book [Bec00]. I only cover a subset of these practices and refer the reader to the article for a good introduction to the method and the book for a good explanation of the philosophy and rationale of the method, in addition to a good introduction to the method.

Testing

According to the article [Bec99], this can be considered the heart of XP. The developers should write automatic tests for their code *prior* to writing code. This helps designing the code so that it is easy to test, and when the test (usually more than one) runs you know that your code is done and you can move on to the next task with confidence that your code works like it should. Before you check in your code you need to make sure that all the tests run. If they don't, then you have broken something and need to fix it first.

In addition to the developers writing tests, XP also specifies that the customers should write tests. These tests are used by the developers to help them understand the requirements better as well as by the customer at the end of an iteration to verify that the functionality delivered matches the requirements specified.

Pair Programming

When I talk to people about XP, their first thought is often "ah, pair programming". I guess the reason for this might be because the idea of pair programming is quite extreme. I imagine it can be quite hard to convince a customer that two programmers working with only one keyboard and monitor is an effective use of resources. However, according to XP it is.

Legend has it that "two heads work better than one". Programming is a complex and difficult job (see for instance the article "No Silver Bullet" in [FPB78]), so having two people cooperate should make this complexity easier to handle and produce a better result. [Bec00, pp100–102]

Another reasons why pair programming is a good idea, is that when people are tired or under stress they will be tempted to skip some of the other practices. Chances are the other person on the pair programming team will object to this and make sure the practices are followed. [Bec00, pp100–102]

Refactoring

According to Erich Gamma [Fow00, xiii] the term was conceived in Smalltalk[2] circles but was soon adopted into other programming camps.

Refactoring is a disciplined way to clean up code and improve design without changing the functionality of the code. Martin Fowler writes this in his book "Refactoring" [Fow00] and states that as code is modified and changed, the design will deteriorate and eventually it will be inviable to maintain the code anymore, unless you practice refactoring.

Since XP is all about changing the code and design when you need it, refactoring is a crucial practice.

The Planning Game

Instead of a lengthy requirements specification phase, XP employs what is called "The Planning Game". At the beginning of the project about a month is spent exploring the requirements and deciding what should be built, and how to build it. During this phase the developers work closely together with the potential users and the customer to learn what is required. A set of use cases is compiled and the developers estimate how much time is needed to implement the individual use cases.

The customer chooses what use cases are most important and should be completed first, and the developers start working on these. At the end of the iteration, the customer is presented with the completed product increment and decides what goes in next.

Small Releases

Like most other agile methods, XP focuses on releasing a small set of valuable features often. Each feature delivered should be complete and tested. In other

[2]The pioneering object-oriented programming system developed in 1972 (Source: The Free On-line Dictionary of Computing).

words, short iterations and valuable functionality delivered after each increment.

Simple Design

As mentioned earlier, software development is complex. Since humans have a limited ability to deal with complexity, XP preaches the KISS[3] principle for design. Do the simplest thing that works today. If you need something more tomorrow, then wait until then before you change the design to accommodate it.

On-site Customer

To make up for the low formality on the requirements, XP says that the team should be able to ask a representative from the customer whenever they need extra information. Thus, if a developer is unsure of whether he has understood the use case he's working on he should be able to discuss the case with the user with as little overhead as possible. That is, stand up and go to the office next door or similar.

Continuous Integration

As mentioned under *testing* the developers should never commit code that doesn't pass the tests. This is related to the practice of continuous integration. The code in the repository should compile and work[4] at all times.

When you are working on your local version of the code, you want to be able to do changes everywhere in the code so that you can practice refactoring. However, this increases the chances of conflicting changes. The practice of continuous integration minimizes this problem since the worst that can happen is that you lose an hour or two of work. In most cases the conflicts will be easy to fix.

[3] Keep It Simple, Stupid. See `http://en.wikipedia.org/wiki/KISS_principle`

[4] all tests pass

2.5 Scrum

Since the project I have studied used a method based on Scrum, I will cover this development method in some detail and thus give it a Section on it's own even though it is one of the agile methods.

According to Ken Schwaber [Sch03, p xvii], one of the originators of the Scrum method, scrum is a process for managing complex projects. He stresses that it isn't just limited to software development. However, software development projects have a tendency to be very complex [FPB78] and so Scrum is well suited for managing them.

2.5.1 Overview

The Scrum method is incremental. Each increment is called *a sprint* and is recommended to last for four weeks. Before the sprint, there is a *sprint planning meeting* where the customer decides what features should be implemented in the upcoming sprint. During the sprint, the team meets daily at a short meeting called a scrum or the *daily stand-up meeting*. At the end of a sprint, a *sprint review meeting* is held where the customer gets to see what was accomplished during the sprint. The team can also hold a *sprint retrospective meeting* where they look at the process and tries to find out what went well and what can be improved.

Schwaber uses Figure 2.5 to visualise the flow of the method. The upper circle represents the daily activities of the team members, while the lower circle represents the development activities that occur during a sprint.

2.5.2 Roles

There are three different roles in a Scrum project; the *customer*, the *team* and the *scrum master*.

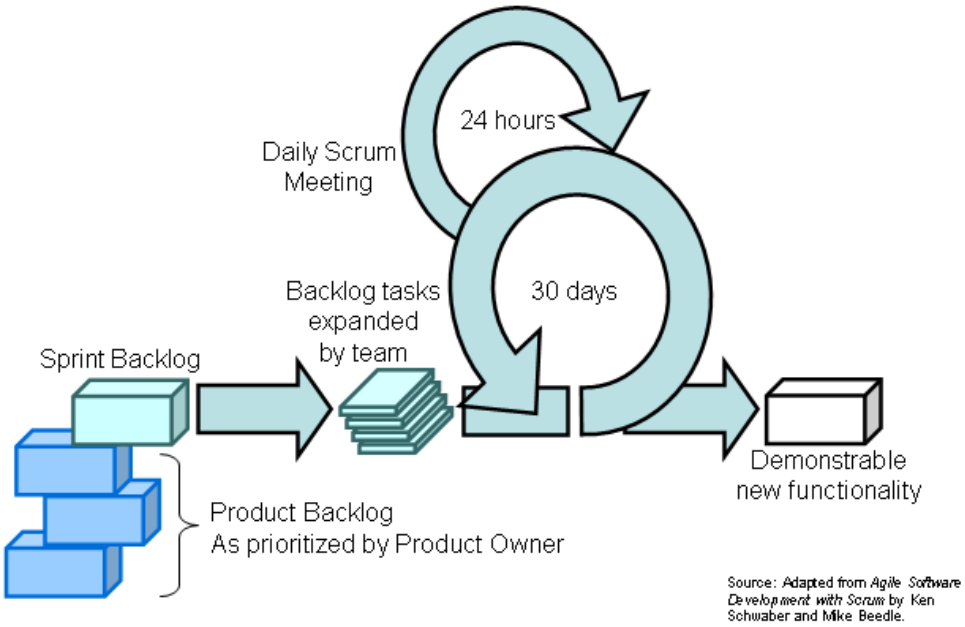

Figure 2.5: Overview of the Scrum method.

The Customer

The customers job is to represent all the stake holders in the project. There can be more than one person in this role, and they are responsible for funding the project as well as creating and prioritizing the list of wanted functionality that should drive the development effort. This list is the *product backlog* and is described further below.

The Team

The team is responsible for developing the functionality requested by the customer. The team is self-managing and responsible for figuring out how to best turn a product backlog into an increment of product within an iteration. They carry the responsibility of the success of each iteration and the project as a whole.

The Scrum Master

Unlike the usual project leader role, where the project leader usually takes the blame when the project gets delayed or don't deliver according to expectations, the scrum master is only responsible for one thing; the Scrum process. His job is to help the customer and the team to understand and apply Scrum to the project. He should help adapt the method to the company culture and make sure that it is being used properly.

He is also responsible for taking care of impediments so that the team can concentrate on actually developing software.

According to Schwaber [Sch03], the Scrum master role isn't necessarily full time and so he can for instance be assigned more than one project or work on the team as a developer.

2.5.3 Scrum Artifacts

Since Scrum is an agile method, it follows that the formality of the project is as low as possible. However, it is considered important that the customer can see the project progress since this improves their motivation and involvement. Also, the team needs some formality to help them cooperate and focus their work.

The artifacts of Scrum are

- the product backlog
- the project burndown chart
- the sprint backlog
- the sprint burndown chart
- the impediments list

The Product Backlog

This could be considered equivalent to the requirements specifications we know from the methods presented above, but there is one big difference. Instead of a long description of each requirement, the product backlog only has a single sentence description of each requirement. This sentence should be enough to remind the customer and the developers of what the feature is.

The Product Backlog is a list of such single sentence requirements. It is *the customers* responsibility to keep it prioritized and updated. The customer adds requirements to this list and then the team is responsible for estimating how long it will take to implement the. An example of a product backlog is shown in Figure 2.6.

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A

Figure 2.6: A product backlog maintained in Microsoft Excel. Figure taken from `www.mountaingoatsoftware.com`.

The Project Burndown Chart

This is a simple two dimensional graph with the work remaining on the Product Backlog as the **y** axis and the time elapsed since project startup as the **x** axis. The graph should give a visual representation of the project speed. It can also

be (and usually is) used to see when the project will be completed at the current development speed as can be seen from Figure 2.7.

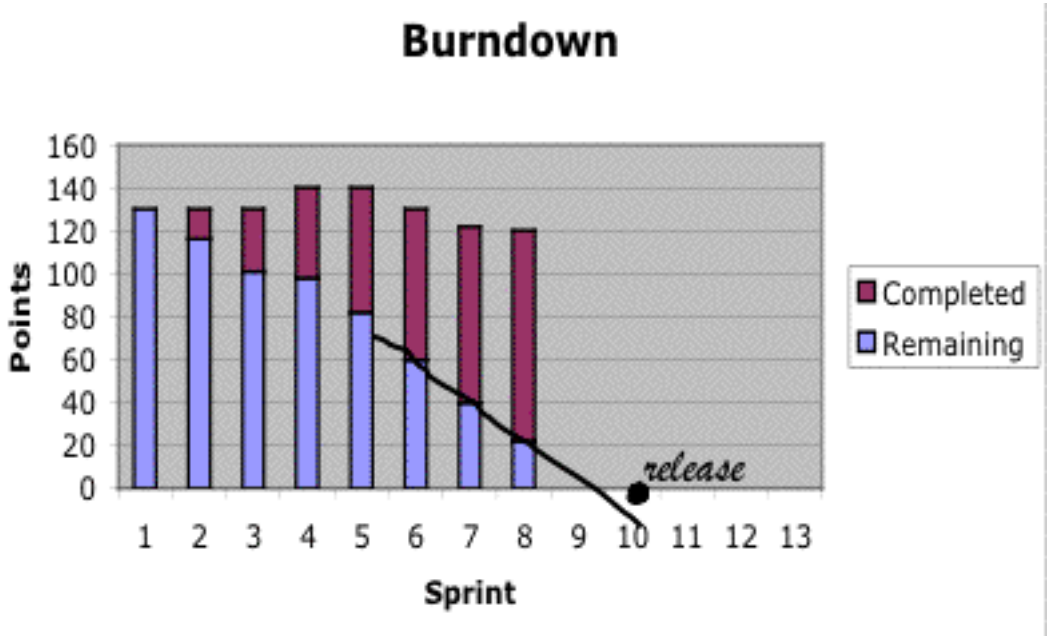

Figure 2.7: Example of a project burndown chart made by Brian Marick, `http://www.testing.com/cgi-bin/blog/2004/10/21`. Notice how the projection of when the project is completed is hand drawn. Marick did this to emphasize that it is only a projection.

The Sprint Backlog

This is a list of tasks maintained and compiled by *the team* based on the items from the product backlog that were selected to be part of the sprint. The list is similar to the product backlog, but there is a big difference. Where the items on the product backlog are features requested by the user, the sprint backlog is a list of tasks the developers must do to implement the items that the customer chose from the product backlog. The customer doesn't need to know about the items on the sprint backlog.

A general rule for the tasks on the sprint backlog is that they should be relatively short, i.e. between one hour and two days. This makes it easier to estimate the

tasks, something that makes the sprint burndown chart (presented below) more accurate.

The Sprint Burndown Chart

This is quite similar to the Project Burndown Chart, only that it measures the progress of the sprint instead of the project.

If all goes well, the sprint burndown chart should look like Figure 2.8 (a). However, in most cases, it looks more like Figure 2.8 (b). This is because the team usually discovers tasks they did not consider but that must be added to the sprint backlog. Since the chart displays the amount of work remaining and not the amount of work completed, the graph can in fact increase from one day to the next.

The burndown chart will in most cases provide ample warning if the team is in over its head and the sprint is too large to complete in time. When this is discovered, the team should discuss the issue with the customer and the customer can choose whether to abort the sprint or decide what backlog item can be moved back into the product backlog. In the last case, the team will end up with a new sprint backlog that they should be able to complete on time. This will be illustrated on the burndown chart by a sudden drop of the graph.

Impediment List

An *impediment* is something that is holding back development in some way or another. As mentioned above, it is the scrum masters responsibility to deal with any such impediments. This list is simply a set of tasks that the scrum master uses to track the impediments that needs to be solved.

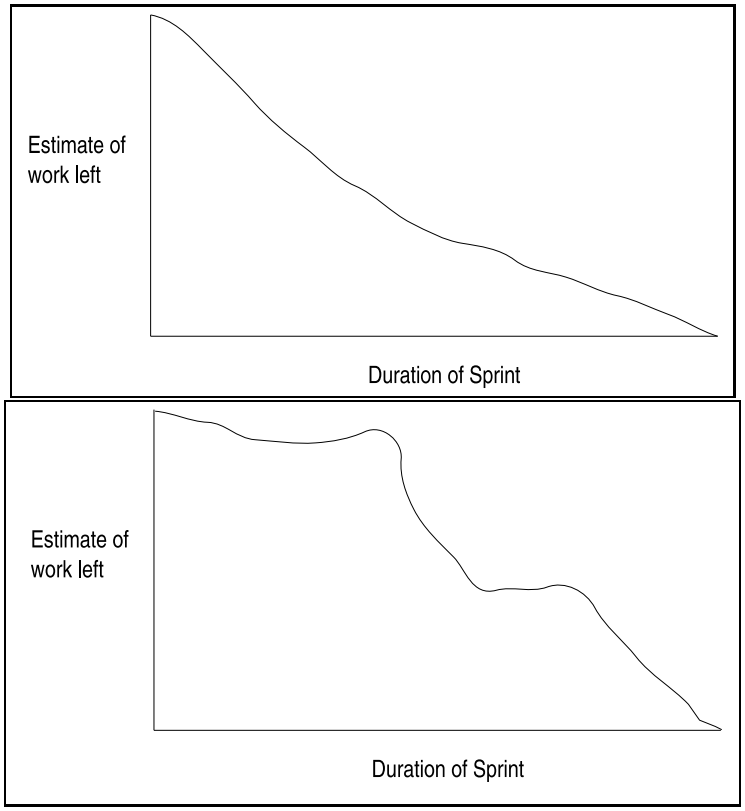

Figure 2.8: Examples of Sprint Burndown Charts used in the Scrum process.

2.5.4 Sprints

As mentioned above, all work is done in sprints lasting four weeks. Each sprint is started with a planning meeting divided in two sessions of at most 4 hours each, as explained below.

How the team works during the sprint is not specified, however, Schwaber has written that XP compliments Scrum nicely [MS02]. XP covers engineering practices but doesn't go into detail on management practices and Scrum doesn't cover engineering practices but is quite clear on management practices. The way I see it, Scrum can be considered a replacement of the planning game in XP.

The Sprint Planning Meeting

In the first session, the Customer chooses high priority items from the product backlog that should be completed in the upcoming Sprint. The customer explains the items to the team and they give an estimate on how long it will take to complete it. The sprint backlog is filled so that the sum of the item estimates is about the same as the available work time of the team during the upcoming sprint.

Here is an example. The customer has selected 5 items from the product backlog to be included in the upcoming sprint. The team discuss these items and decide what needs to be done to complete them. This will result in an initial set of tasks for each item that together constitute the sprint backlog. I say initial since the developers will surely discover new tasks during the sprint that must also be included in the sprint backlog.

In the second session, the team breaks the selected backlog items into smaller work sized tasks that are inserted into the sprint backlog. The customer should be available for questions and clarifications during this process.

The Daily Activities

During the sprint, the developers work on the items in the sprint backlog. Every day the developers synchronize their progress in a daily Scrum meeting that should last no longer than 15 minutes. During the meeting, all the developers will tell the others what they did since the last Scrum, if there are any impediments obstructing their work and what they are planning on doing until the next Scrum.

Another important day to day activity is updating the sprint backlog and burndown chart.

Sprint Review Meeting

At the end of the sprint, the team meets with the customer and presents the result of the sprint. The users demonstrates the functionality they have completed and

get feedback from the customer.

If the demonstrated functionality is what the customer wanted then this gives the team a feeling of accomplishment as well as the customer a proof that the project is moving in the right direction. If the demonstrated functionality isn't quite what the customer was looking for it is now easy to explain how it is different and what should be done next. In some cases it is enough to make a few changes while in other cases the implemented functionality must be discarded.

Sprint Retrospect Meeting

The intention of this meeting is to help the team improve their development process. The meeting is attended by the team, the scrum master and the customer (optional). During the meeting the team members take turns saying what went well during the last sprint, and what could be improved. After all team members have had their say, they prioritize the possible improvements and discuss them in order. The meeting should not last more than 3 hours.

2.5.5 Project Startup

Ken Schwaber has had much success with his kick-starting of Scrum projects as described in the book Agile Project Management with Scrum [Sch03]. This process goes as follows.

The Scrum Master works with the customer and prepares a backlog. Then the Scrum Master, the Customer and the Team uses one day to go over this backlog. During this first day the customer explains the items in the backlog to the team, and the team estimates how much work it would take to implement this. The customer then prioritizes the items in the backlog and divides the backlog items into sprints.

The following day is the first day of the first sprint. This first sprint isn't very different from the following sprints, except that the first part of the sprint planning meeting has already been completed.

The team is now in complete control and have one task, namely to deliver the functionality the customer has requested. The sprint has begun.

2.5.6 Project Completion

As the project moves on and sprints are being completed, the customer will receive increments of the product. If the customer realizes that the product is good enough and that further development is unnecessary, then he should be able to stop the project. Depending on the contract that has been negotiated, there can be a penalty fee for premature termination of the project.

Chapter 3

Case Study

To discover whether an agile approach works well in a real world setting, I conducted a case study of a project Company 1 is running with a team of software developers from Company 2. See Appendix A for a thorough description of how the case study was conducted. In this chapter I present the results of my study.

I start out by presenting the involved parties and the history of the project. Then I write about how and why they adopted the new method. Following this I describe the development method I observed. I complete the chapter with a description of what might happen to the project and the team in the future.

3.1 The Actors

A Software Development project usually has a number of involved parties, or actors as they are sometimes called.

In this case we have *Company 1*, who saw the need for the software; *Company 2*, the consulting firm that was hired by Company 1 to implement the software; *the team* of developers who were sent by Company 2 to develop the requested software and finally *the customer*, a group of people who work for Company 1 and is responsible for the project.

3.1.1 Company 1

Company 1 is a very large company responsible for planning, building and maintaining roads. They are also responsible for the supervision of cars, trucks and other road-users.

3.1.2 Company 2

Company 2 is a Norwegian consultant firm who specializes in software development and object oriented programming. The employees are mostly highly educated and the firm have a reputation of only employing skilled developers with at least three years of past programming experience.

The firm was established in 1999 and have since grown to become 70 employees (source Developer 1).

3.1.3 The Team

The development team from Company 2 have grown and shrunk during the project period. It started out with only one developer, then it grew to six people for a while. For a summary of the project history, see Section 3.2.2. To keep the involved developers anonymous I will call them Developer 1 to 8.

Developer 1 was the first developer on the project and is still working on it. Developer 7 and 8 worked on the project in 2003, but were moved to different projects at the end of the year. Developer 2 joined the team in August 2004 and have been working on it since then. Developer 3 and 4 joined the project in September 2005 and Developer 5 and 6 joined in December the same year. In May 2006 Developer 4 and 6 were moved to a different project.

Today the team consists of four developers, three male and one female (Developer 5), working full time on the project. The developers work at Company 1 in two adjacent offices in a floor dedicated to consulting firms working on projects for

Company 1. The offices are big enough that two people can work next to each other on different tasks, which is what the team were doing when I observed them.

From my observations, they seem to be a friendly group that respect each other and have fun together. My interviews give the same impression, for example in the interview with Developer 5 when I asked if she felt accepted by the other developers she answered “Absolutely. They are forthcoming and helpful...”. Earlier in the interview she asserted that she was very glad she got to work in that team.

3.1.4 The Customer

While it can be argued that Company 1 is the customer, I choose to use this term to describe the people from Company 1 who is responsible for the project and who interact with the team. From my interview with *Customer 1*, there are three people from the company who make up this group. I call them Customer 1 to 3. Customer 1 is the person who is most involved with the team, Customer 2 is the project leader and Customer 3 is the person who cooperates with Customer 1 to prioritize and maintain the product backlog.

3.2 The Project

The project started in 2002 and consisted of developing an electronic replacement for the vehicle control process. This process is currently based on filling out and handling forms, something that is both time consuming and error prone.

I present a brief overview of the product to give the reader an impression of what the team was working on, and then I summarize the history of the project as presented to me by Leader 1 and Developer 1 from Company 2.

3.2.1 The Product

The product Company 2 was hired to develop was a complete solution for vehicle control. The system should support vehicle controls performed in a hall for scheduled controls, or by the road side for sampling tests.

Some of the problems Company 1 had with the old system, was that it was hard to follow up and check if vehicles that didn't pass the control were fixed after the control. Vehicles driving long distances were sometimes stopped and checked several times during the trip.

The solution Company 2 is developing consists of a PDA[1] with software for conducting the control, a web interface for scheduling controls and server software for storing and managing the control data as well as for connecting the system to various databases for looking up information regarding the vehicles and drivers.

The user interface for conducting controls was designed so that the user should be able to do as little typing as possible. One of the ways this was achieved was by utilizing a bar code scanner on the PDA. The user scans the bar codes on the vehicle license plate and driver license, upon which the system retrieves information regarding this vehicle and driver from the central server over a wireless network.

When the user has registered all necessary data, he proceeds with filling in the results of one of several pre-defined tests.

After the control is completed, a receipt is printed on a printer in the controllers car and the data is sent to the server for storage and later follow up action. The printing of the receipt is also done using the network, so the controller never has to connect the PDA to any other device during the control.

[1] Personal Digital Assistant, aka handheld computer. A small computer the size of a calculator. Usually has a touchscreen and can be considered the technological equivalent of a sixth sense or Filofax.

3.2.2 History

The project was started in August 2002 when Developer 1 and Leader 1 worked out the requirements specification. This work was completed at the end of November or the beginning of December the same year and given to Developer 7 and Developer 8. They created a proposed solution which was approved January 2003.

The project was staffed with a development team consisting of Developer 1, Developer 7 and Developer 8. They delivered a pilot version for testing in August 2003 after which Developer 8 was moved to another project. Testing and bug fixing continued into September.

In September Developer 7 was moved to another project as well. Developer 1 continued working on improvements and changes to the system as well as training the users. The system was supposed to be tested during the autumn of 2003 and then put into test service at the start of 2004, but in January 2004 the infrastructure was proved to be unstable and the release was postponed. Among other things, the GPRS[2] network did not work well enough to maintain the VPN[3] connection that was required to access the database with information about drivers and vehicles. The VPN solution was discarded during the Summer of 2004 and it was decided that a different security scheme had to be developed.

Developer 1 continued the system development during Spring 2004 while waiting for the network problems to be fixed. During this period the functionality and ambitions of the project increased.

In August 2004 Developer 2 was added to the project to help add functionality and improve the code quality. Development of an SSL[4] authentication solution to replace the discarded VPN solution was started.

The product was put into test service in January 2005. Pilot testing was conducted and finally terminated in June 2005 due to problems with the service environment; the GPRS network and the main portal infrastructure were unstable. During this

[2]General Packet Radio Service, used for data communication over the mobile phone network

[3]Virtual Private Network, used to provide a secure channel over an insecure network

[4]Secure Sockets Layer, used for authentication and encryption of web pages and many other internet services.

period there were several releases of the product, and the final release before the summer was considered functionally stable (e.g. all necessary functionality was present). It took until December 2005 for the main portal infrastructure to become fully operational. The GPRS infrastructure wasn't reported stable until February 2006.

In August 2005 a refactoring of large parts of the data model and user interface was initiated. This resulted in a new release candidate November 2005.

Developer 3 and Developer 4 were added to the project in September 2005 to implement new features. The scope of the product was increased to support digital tachographs, a system for keeping track of how long truck drivers have been driving without taking breaks[5].

The Scrum method for project management was adopted in October or November 2005 to improve coordination of the different project activities as a result of the project growth. This is covered more thoroughly in Section 3.3.

After a presentation of the product on Iceland, December 2005, an initiative by Company 2 to internationalize the product was started. Developer 5 and Developer 6 were employed and put to work on this task. They started using the Scrum method and took part in the Sprint planning meetings of the main development group in February 2006.

I requested permission to study the project and, after a few phone calls and emails, was invited to visit Company 2 in March. This visit is described in detail in Appendix A.

The internationalization work was completed in May 2006, freeing two developers. Developer 4 and Developer 6 were moved to different projects while Developer 5 replaced Developer 4 in the main development group.

I made another visit in May (see Appendix A for details). The project was, according to Customer 1, proceeding as planned, and they will hopefully start using the product in September or October 2006.

[5] I have not looked into the details, but there are regulations on how much long a truck driver can drive without resting.

3.3 Method Adoption

As can be seen from Section 3.2.2, Scrum was adopted near the end of 2005 as an effort to improve the coordination of the different project activities.

In the interview with Developer 1, he said that while they were two developers it was easy to coordinate their work. When the team grew to four developers, they worked for a while as two teams. After a while this lead to conflicting and overlapping changes, so they decided to reorganize themselves into one team.

3.3.1 Why Base the Method on Scrum?

Prior to the implementation of Scrum, one of the other development teams working for Company 2 reported good results from implementing Scrum in their project. Agile methods and Scrum had been discussed at staff meetings at Company 2, so the method was known to the developers of the project and they could talk to coworkers who had first hand experience with this method.

As a result of the above, the team decided to adopt scrum, but at their own pace and premises instead of uncritically by the book. (Source: Interview with Developer 1).

3.3.2 The Adoption

From my interviews I learned that the team had a presentation of the Scrum development method from another employee at Company 2. This presentation lasted for about an hour and was a presentation of how the method had worked for another team from Company 2.

In addition to this presentation, Developer 1 read the article “Agile Project Management with Scrum” by Dafydd Rees [Ree04] that described the Scrum method. The other team members that I talked to had not read any documents about Scrum.

Before they adopted the new method, they maintained a list of development tasks

in a spreadsheet. This list of tasks was moved to their product backlog which they maintain in a tool called JIRA[6].

3.4 The Development Method

Based on observations, questions and interviews, I have constructed the following description of the development method the team is using today.

3.4.1 Overview

It is an iterative and incremental method with iterations lasting 2 weeks and releases done at least once every 6 months but with no set interval as far as I could tell. The method is inspired by the Scrum method described in Section 2.5, but there are some differences which I will point out in Chapter 5.

Every iteration, or sprint as the team calls it, starts out with a planning meeting. After this meeting the team has a concrete list of tasks that should be completed by the end of the sprint. The developers show up at work between 7 and 9. At 10 they get together for a daily stand-up meeting, after which they continue working on their tasks.

3.4.2 The Roles

During my observation I noticed two different roles on the project. The team and the customer. These were modelled after the roles in Scrum, and were similar enough that a description would be a repetition of what I wrote in Section 2.5.2.

In the interviews and discussions the team said that Developer 1 also had the Scrum Master role, however I did not have a chance to observe this during my visits.

[6]See `http://www.atlassian.com/software/jira/` for more information on this tool.

3.4.3 The Backlogs

The team had two backlogs, or lists of tasks; the product backlog and the sprint backlog. The first was a list of tasks of different sizes and importance that the team should work on to complete the product. The list had approximately 150 tasks when I conducted my study. The second was a list of tasks, selected from the product backlog during the sprint planning meeting, that should be completed during the current sprint.

The sprint backlog was initially a subset of the product backlog, but it often changed during the sprint as new tasks were discovered and added. This reflects what is written in Section 2.5.

When the customer or users found errors or missing features they wanted to add to the product, they sent an email to the developers (through the customer or a mailinglist). These requests were added to the product backlog by the team and the customer were then responsible for assigning a priority to the item. (Source: Interview with Customer 1).

During a discussion about the product backlog I learned that the idea behind the product backlog that was used on this project was very different from the one used in Scrum. The goal in Scrum is to complete all the items in the product backlog, eventually delivering a finished product to the user and ending the development[7]. The team didn't agree with this way to treat the product backlog. Instead, the product backlog would always have many tasks. There is always something that needs to be done on a software system. It can be adding features, improving features, fixing features, improving design, etc. Because of this, the team did not aim to end up with an empty backlog. Instead, they expected the backlog to reflect how mature the project was. A very mature project will have many bugfixing and maintenance tasks, while an immature project will have more feature tasks. To paraphrase Developer 1

> An empty backlog means that the project is dead

[7] this is not necessarily how it is done, but that was how we talked about it during this discussion

3.4.4 Priority-Assignment Meeting

While I have not been able to observe one of these meetings, both Customer 1 and Developer 1 informs me that they meet at every sprint to discuss the product backlog and assign priorities. For practical reasons there is no regular day and time for this meeting.

During this meeting they assign priorities to all new feature requests and bug reports from the users. They discuss the progress of the project and the direction of the development.

3.4.5 Sprint Pre-Planning Meetings

Thursday or Friday at the end of the sprint the team has a meeting right after the stand-up meeting. Unfortunately I wasn't able to observe one of these meetings, so my knowledge about this meeting is scarce. What I do know I got in an email from Developer 1. He says that during this meeting they define a goal for the next sprint, and they do some coarse planning. The goals are reached in dialogue with the customer. If the customer needs a bugfix release, they set this as their goal. Often the goal is obvious, for instance to do a new release, to add support for a new important feature, etc. If any of the developers have tasks they wish to include in the sprint, they can bring this up at the meeting and do a vote on whether to include them or not.

It seems that this is also the meeting where they adapt their method by including new practices (from for example Scrum or XP). They also discuss their current practices and decide whether to keep or discard them.

At the end of this meeting, the top-priority tasks are divided between the developers. They then have to present these tasks during the group estimation process at the sprint planning meeting.

3.4.6 Sprint Planning Meetings

Before the sprint planning meeting, the tasks with the highest priorities were distributed among the team members who prepared a short presentation of what the tasks implied and how they could be accomplished. The tasks should also be estimated and checked if they were duplicates or made obsolete or affected by earlier changes.

At the meeting, the team members presented their list of task and discussed the tasks that weren't completely clear. They seemed to cooperate very well in selecting tasks and balancing the amount of work they scheduled for the coming sprint with the amount of work they would be doing in the sprint. At the end of the meeting when they were doing selecting tasks, they quickly decided who would update the software system they used for keeping track of the task, as well as their documentation system[8].

During my study, the team adopted the *planning poker* practice as described in Section 3.4.8. This changed the meeting to a large degree. Now the meetings were longer and the tasks were discussed in more detail than earlier. To make this possible they also distributed the tasks that would be considered for the upcoming sprint among the team members prior to the meeting. They then had to present their task and explain why it should be part of the sprint, or why it could wait and so on.

3.4.7 Sprints

During the Sprint, tasks were worked on, completed and moved from the sprint backlog list to a list of items for testing and then on to a list of completed items when the test had been run and approved by a different developer than the one who wrote the code.

Because my study focused mostly on the development method and not the coding

[8]I have not written about these systems since I didn't consider them very important to the development method.

practices, I don't cover how they worked with the code from day to day. However, I did notice that the developers talked to each other and at times moved between the offices to discuss the code with one of the other developers.

The Stand-Up Meetings

The *stand-up meetings* were held around a tall table in a cafeteria in the ground floor of the building. On my first visit, the meetings were held at 09:00. On my second visit they were held at 10:00. They started, when everybody arrived, with one of the team members telling what they did yesterday and what they were planning on doing today. Then, clockwise, everybody did the same.

I noticed that the landscape in the cafeteria was open and at times noisy, but this didn't seem to bother the team. At times I had a hard time picking up what was said, but this was probably because I stood a bit outside of the circle and they did not talk directly at me but with each other. When I asked why the meetings were held in the cafeteria, I was told it was because the coffee there was better than the one near their offices.

Once or twice when I was there, someone brought up an impediment which was keeping them from completing their task. An example of how one of these impediments were dealt with is when Developer 5 was unsure of how the design of the network communication protocol worked. This made it hard for her to implement the task she was working on. After the stand-up meeting the team members held a design meeting where they explained this design. While the meeting was primarily for Developer 5, I got the impression that the other developers also learned a bit from the meeting as they had to consult the source code several times during this presentation.

The Burndown Chart

The Burndown Chart gave a visual representation of this, as descibed in Section 2.5.

3.4.8 Planning Poker

One of the problems with estimating tasks is that it is hard to estimate correctly, since it is easy to overlook subtasks. One way to fix this is to use group estimation where the tasks are evaluated by a group. A problem with this is that often the developer who knows the code best is quickest with his estimate, and when an estimate is given the other developers trust that the estimate is good.

The way planning poker works, is similar to group estimation. The difference is that now all the developers must estimate the task and write down their estimate. When everyone is ready, the players show their estimate at the same time. [Gre02]

Now the person with the highest estimate and the person with the lowest estimate must explain why they believe their estimate is correct. After a short discussion, everybody usually have a better understanding of the task, the estimate is often better than using regular discussion, and finally all the developers must join in the estimation. This gives the junior developers more practice since they might otherwise keep quiet during regular estimation. [Gre02], [Coh05, pp56–57].

3.4.9 Project Completion

I have not been able to observe how the project ends. I have not asked questions about this to the developers either, but I did talk about what will happen next in my interview with Customer 1. He says that the product will hopefully be shipped in September or October this year.

After release, the product will be actively maintained and updated for years. To paraphrase the customer

> I consider this a life time project. If you deliver a product and say “here you go” and don’t maintain it properly then it will be up to date for a few months before you have to say “good-bye”. The rules change all the time; we get a “bible” with 1200–1500 pages every other year, so there are a lot of changes.

Customer 1 did not know whether the current development team will be kept during maintenance, but he thought there might be some changes since the project will be moved to a different part of the organization.

Chapter 4

Research Method

As described in the introduction, the objective of this study was to learn more about how agile software development is practiced in Norway. Through talks with my councilor we concluded that the most appropriate research method for me would be an interpretive case study.

I start this chapter with a section on research method theory. Following this I give an overview of what I have done to collect data for my study. I discuss the validity of my data and to conclude the chapter I do a post-hoc analysis of my research using the seven principles that Klein & Meyers present in their article [KM99].

4.1 Theory

Robert Galliers [Gal94] roughly divides information system (IS) research into two groups, the scientific and the interpretive. The former is mostly used when the object of the study can be observed objectively and the scientist can generalize from the study. These studies usually involves proving or disproving a hypothesis that is decided upon before the research is started. Interpretive research methods take into consideration that what is observed is also interpreted by the observer, and that this interpretation will influence the research.

Chen and Hirschheim [CH04] has done a literature study of 1893 articles released between 1991 and 2001. Their research showed that 81% of all IS research is done using traditional scientific methods. Galliers and Frank Land [GL87] pointed out this trend in 1987 and argued that there is a tendency to favor the traditional research methods even when interpretive methods would give better results. They say that for IS research it is often appropriate to include behavioral and organizational considerations. While this increases the complexity and decreases the precision of scientific studies, it goes well with the interpretive research methods.

My research is interpretive. I have conducted a case study where I have observed, interviewed and gathered information that I have structured and interpreted in this paper. As a result, this paper is far from objective. I have tried to make my preconceived notions as transparent to the reader as possible by including the story of how the study started (Section 1) and how I performed my research (Appendix A).

In a further attempt to increase the quality of my research, I have done a post-hoc analysis of it using the seven principles that Klein & Meyers present in their article [KM99].

4.2 The Study

One of the biggest problems with conducting empirical research is that the researcher is biased. If he/she didn't already have an opinion on or an interest in the subject matter why research it?

Before I started my study I believed (and I still do) that agile methods are better than the sequential methods I knew about from earlier in my studies. Yes, there are scenarios where agile methods are impractical, but in small to medium sized teams with non-critical (loss of money, not life if the software fails) development projects I believe agile is the way to go.

The data collection was done using semi-structured interviews, observations from meetings I participated in and of course general observations made while I was

Date	Object	Details
2006-03-17	Developer 6 (Junior)	A semi-structured interview about his experience with joining the project and the development method.
2006-05-23	Developer 1 (Senior)	A semi-structured interview about the project, development method, etc.
2006-05-23	Developer 2 (Senior)	A semi-structured interview about the project, development method, etc.
2006-05-24	Developer 3 (Senior)	A semi-structured interview about the project, development method, etc.
2006-05-24	Developer 5 (Junior)	A semi-structured interview about the project, development method, etc.
2006-06-12	Customer	A semi-structured interview about the project, development method (as seen by the customer), the customers experiences, etc.

Table 4.1: An overview over the different interviews I performed during the study.

visiting Company 2. In addition I've received some of my data through email correspondence with Leader 1, Developer 6 and Developer 1.

During my study I did two visits to observe and interact with the development team. The first visit was from March 16th and lasted until March 21st. The second visit was from May 21st and lasted until May 25th. For a detailed rendition of these visits, see Appendix A. To give an overview I've summarized the interviews in Table 4.1, the meetings I attended in Table 4.2 and other sources in Table 4.3.

4.3 Validity of my Research Data

As mentioned in Appendix A, I didn't do a good job at taking notes during my second visit. Because of this I had to trust my memory to some degree when I reconstructed the sequence of events during the study. So it is prudent to ask whether this has affected the validity of my research data.

For very detailed information like the length of meetings, my numbers are approximations and should not be considered very accurate. When I did my observations

Date and Length	Place	Participants	Details
2006-03-16 2h40m	Company 2	Leader 1, Senior 2, 2 researchers from SINTEF	Discussion of cooperation between SINTEF and Company 1, introduction to the company and project.
2006-03-16 3h	Company 2	Employees of Company 2	Staff meeting. Mostly unrelated to the project.
2006-03-16 2h	Company 2	About 10 developers from Company 2	Interest group on Ruby on Rails. Unrelated to the project.
2006-03-17 1h10m	Company 2	Leader 1 and Developer 6	Presentation of the project.
2006-03-17 1h	Company 2	Developer 6	More information on the project, questions and answers.
2006-03-18 4h	Company 1	Developers 1-4, 6 and Senior 1	Sprint Planning Meeting and discussion of development methods and practices.
2006-03-19 15m	Company 1	Developers 1-4	Stand-up Meeting
2006-05-22 15m	Company 1	Developers 1,3,5	Stand-up Meeting
2006-05-22 4h	Company 1	Developers 1-3,5	Sprint Planning Meeting
2006-05-23 15m	Company 1	Developers 1-3,5	Stand-up Meeting
2006-05-24 15m	Company 1	Developers 1-3,5	Stand-up Meeting

Table 4.2: An overview over the different meetings I attended during the study.

Source	Type	Details
Leader 1	Email	History of the project
Developer 1	Email	History of the project
Developer 1	Email	Summary of the development method
Developer 6	Email	Introduction to the project
Developer 1	SMS[1]	Answers to short questions

Table 4.3: An overview over other important sources for information during the study.

I did not focus much on details; instead I wanted to concentrate on the big picture. Here I have been more thorough. When I have been in doubt about my observations, I have either sought assistance from one of the Developers or I have decided not to include that piece of information.

Because of this I believe that my research data is valid.

4.4 Analysis of My Work

To judge the quality of interpretive research, Klein & Meyers [KM99] has presented 7 principles they believe should be used to validate interpreted case studies. I've done a post-hoc analysis of my research using these principles.

4.4.1 The Fundamental Principle of Hermeneutic Circle

To quote Klein & Meyers,

> This principle suggests that all human understanding is achieved by iterating between considering the interdependent meaning of parts and the whole that they form.

What this means can be made clearer with an example. Consider the sentence "they are playing football". For most Norwegians, this sentence would give an image of people running around on a field while trying to kick a round ball into a goal. On the other hand, most Americans would imagine people on a field trying

to get an egg shaped ball over a line or a mark. Due to a different interpretation of the word football, the word playing gets a different meaning. Extra information could change the meaning further, for instance if it is clear from the context that "they" are not engaged in sports at all, then the meaning must be metaphorical. They might be throwing ideas back and forth...

This principle is the overreaching principle that the following six principles are built on, and it is used to make sure that there are no gaps or unresolved contradictions in the research material.

Since I can find no gaps or contradictions in my research I believe that by fulfilling the other principles I have also fulfilled this principle.

4.4.2 The Principle of Contextualization

As explained above, the contextual information is paramount in how you interpret the data. To be able to understand the current situation, we need to know the historical and political context.

I have done my best to give a historical context in Section 3.2.2. This was reconstructed from emails and conversations, so it is all secondhand information. However, I see no reason why I would be given false information regarding the projects history. My two main sources for the history chapter was Leader 1 and Developer 1. However, the information I gathered from Customer 1 did not conflict with this, so in conclusion I believe my contextual history is sound.

I have not dug as deep into the political context between Company 1 and Company 2. However, since my study is focused on the development team and their interaction with each other and the customer, this doesn't seem to be relevant. I could have looked more into the economical context of Company 2, since my interview with Customer 1 shows that the resources Company 2 spent on the project were less than Customer 1 thought necessary. However, I don't consider this very relevant for the study.

4.4.3 The Principle of Interaction Between the Researchers and the Subjects

This principle states that the researcher should reflect upon how the research data has been constructed through the interaction between the researchers and the participants.

As I show in Section 4.2, the development team did not seem to have any problems with having me around. They were helpful and gave me all the information I asked for. The only situations where I felt that my presence might have caused the team to act different from normal was during lunch. I would have expected more "shop talk", but found that the discussions seldom went into detail about the design and code they were working with. I suspect that this might have been because I was there, but it might also have been the normal behaviour.

When I was just observing, I had the impression that the team didn't really notice that I was there. During interviews and talks I tried to avoid arguing with their opinions and rather encourage them to explain their point of view. While I suspect that I wasn't always successful at this, I believe the majority of my data is untainted by it.

4.4.4 The Principle of Abstraction and Generalization

I have not done any abstractions or generalizations, so this principle doesn't apply to my research.

4.4.5 The Principle of Dialogical Reasoning

This principle states that the researchers might have pre-conceptions about what he expects to find, and that this can affect the resulting data or leads him/her to revise his/her pre-conceptions. There can be several cycles of this.

In Section 4.2 I state my bias for the reader, so the reader is aware of my pre-conceptions. Further, while I originally believed that the team had a lot to learn

about agile methods as a result of finding out that they hadn't read any literature on Scrum, I revised my opinion based on the interview with the customer and realized that the team were in fact doing a type of agile development and that my original pre-conception was flawed.

4.4.6 The Principle of Multiple Interpretations

This principle doesn't apply to my research since I only have one narrative and my research shows no signs of a possibility for multiple interpretations.

4.4.7 The Principle of Suspicion

In this final principle, Klein & Meyers require me to be sensitive to possible biases and systematic "distortions" in the narratives collected from the participants.

What I need to ask myself as a result of this principle is if what I have been told is the truth or if it might be distorted by the customer or the team to make them look better? (Or for some other reason.) I need to be suspicious of the data I'm fed and look for inconsistencies and indications that what I'm told is exaggerated or similar.

It is of course possible that the project members, upon hearing of my arrival and study, agreed upon a story to make them self look better. However, I consider this unrealistic. Since I have done interviews with 5 team members and nothing I have heard have made me aware of conflicting informations, and given that what the customer told me in his interview fit with what I had learned in earlier interviews, I believe that what I have been told is true.

When I constructed the history of the project, I sent an email to Developer 1 who had been on the project since the beginning. However, for some reason the email got lost and it took two weeks before he replied. During this time I had sent a request to, and received this data from Leader 1. I now had the same information from two separate sources that did not know (at least, I didn't tell them) that I

received the same information from the other source. While the data I received focused on different details, there were no conflicting data.

Chapter 5

Analysis

I start out my analysis by looking at the differences between the Scrum method and the method employed by the team. Keep in mind that the team did not aim to implement the entire Scrum method at once (see Section 3.3.1), so some of the Scrum practices are yet to be implemented.

When I have established some of the differences, I will try to explain the reason for the differences. Finally I conclude the chapter with a few suggestions on how the team can improve their development method.

5.1 Observed differences

During my study, I realized that there were quite a few differences between the method employed at Company 2 and the Scrum method as I understood it from the book [Sch03]. See my description of Scrum in Section 2.5 as well as my description of the method I observed in Section 3.4 for a reminder of how I see the two methods.

5.1.1 Project Startup

I haven't read any books or articles on how to adopt scrum in mid project, so it is hard to say whether the team did this "by the book" or not. I suspect that the way the team simply used their existing task list as the initial product backlog might not be the "right way", but it is obvious that this made the transition to the new method easier.

The alternative would probably be to sit down with the customer and create a completely new product backlog. It would probably take about a day to create the product backlog this way; e.g. 8 hours more than what the team spent on this.

The biggest difference with these two methods must be how the easy solution makes the method change less intruding for the customer, since the alternative requires that the team introduce him to the Scrum method and convince him that this is not a waste of time.

5.1.2 The Roles

The team and the customer worked pretty much as I expected (except for the sprint planning meeting as described below).

The scrum master role seems crucial in projects where the team members and customers are new to the method. At least if the goal is to use Scrum by the book.

The acting scrum master in this project wasn't trained to do this job, and therefore it can't be expected of him to be able to carry it out correctly. Since the team did not aim to implement Scrum by the book, having a proper scrum master might not be necessary.

5.1.3 The Product Backlog

A quick look at the *product backlog* shows a long list of tasks. When talking with the team, it became clear that their understanding of the product backlog was quite

different from the one in Scrum. The team said that the product backlog wasn't supposed to become empty. If it did, that would mean that the project was dead. In Scrum, the goal is to complete all the items in the backlog.

Another difference is that in Scrum, the project backlog is a list of functional and non-functional requirements made by the customer. He should be able to read the backlog and easily assign and change the priorities of the items (i.e. the items should be written in a language understandable by the customer). The items on the teams product backlog were smaller and much more technical. They were mostly written for the developers, not the customer.

A consequence of the last difference is that sprint review meetings will be affected. While the team didn't perform sprint review meetings during my visits, the practice might be adopted later. As described in Section 2.5.4, the user should be able to tell whether the team delivered the promised backlog items or not. With the current backlog this would be quite hard since many of the tasks are technical and have to do with the design and internals of the program.

5.1.4 The Sprint Backlog

I didn't observe any differences between the sprint backlog used by the team and the one described in the literature.

5.1.5 Sprint Planning Meeting

The customer did not attend the sprint planning meetings. In the interview with Customer 1 it becomes clear that the reason for this is that they have too many things to do, so the project gets less attention than it should have. However, the interview also indicates that such a meeting could be possible if the team insisted on it.

The first part of the sprint planning meeting is not done on this project. Instead, there is a separate priority-assignment meeting as described in Section 3.4.4.

Customer 1 was also available for questions and clarifications whenever the team needed this and provided a single point of contact, making it easier for the team to get the information they needed.

In addition he was active in facilitating field testing as well as performing planned testing of the product.

As covered in Section 2.5, these meetings are divided in two distinct sessions. The first with the customer; the second for the team. Each session is not to last for longer than four hours, and after the first session the sprint is considered started.

The planning meetings I attended did not resemble this description. The customer did not attend and consequently part one could not be done as described even if they wanted to. The second part of the meeting was closer, but this part was also different from what I described in Section 2.5.

Instead of splitting the items from the product backlog into tasks for the sprint backlog, tasks from the product backlog were re-estimated and a subset was chosen as the sprint backlog.

5.1.6 Daily Stand-up Meetings

I didn't observe any differences here.

5.2 Reasons for the differences

Why doesn't the Company 2 team follow the Scrum method by the book? Is it only because they haven't read it, or are there other reasons that makes their situation better suited for the solutions they have found?

An easy answer would be that the customer wasn't involved enough to actually perform the necessary practices, however my study seems to indicate that there was in fact a rather good customer involvement.

I'll look into each of the roles, practices and artifacts that I've covered and finally I'll try to make a summary and conclusion.

5.2.1 Backlog

As I covered in 3.4.3, the team moved their list of tasks directly into the product backlog when they adopted the backlog from Scrum. This set the standard for backlog items and this easy solution combined with their understanding of the backlog when they created it is the cause for the difference between Scrums product backlog and the teams product backlog.

Since the team experienced positive effects of the new way to work with a product and sprint backlog, they were happy with it and didn't see any reason to believe that they were doing it wrong.

I believe the impact of this difference is big. First and foremost, the backlog is supposed to belong to the customer, not the team. Giving the customer a backlog that he understands and can identify with makes it easier for him to make changes to it and select what should be worked on next. The current backlog is quite technical in addition to being very long. This makes it infeasible for the customer to go over it in its entirety and change the priorities very often.

The current practice is that the customer prioritizes new items that are added to the backlog, while existing backlog items are left as they are. While this gives the customer some power over what should be worked on, it does not come close to what it should be.

5.2.2 Sprints

I wrote above that the length of the sprint should be four weeks. In an interview, Ken Schwaber [Con06] argues against sprints lasting longer or shorter than this. He asserts that four weeks is the longest period of time that the Customer will be able to wait and still feel involved in the project, while it is the shortest amount of time that a team can deliver a decent amount of valuable functionality.

However, my research shows that the team were happy with two week iterations and that their attempt to use three week iterations had proved inefficient. My interview with Developer 3 shows that when they tried three weeks they lost focus and ended up with with a feeling that they didn't accomplish more than they would during a two week iteration. This lead them to returning to the two week iteration.

I believe the reason for this difference is related to the product backlog. Since the tasks in the product backlog are smaller, I figure the sprints end up smaller as well. This is because with "Scrum sized" backlog items, the team would select maybe between 5 and 10 features to work on. These would be divided into a sprint backlog and work would ensue. As the team works, they will encounter small victories as they can see that their work on the tasks lead to product backlog items being completed. That is, they have two levels of completion; small items for day to day work, and product backlog items once or twice a week which shows progress and provides inspiration.

Since the team only have the small items, they do the day to day work, but they don't get the pleasure of seeing the product backlog items (or functionality) working. This makes it harder to keep the focus over longer periods of time.

5.2.3 Sprint Planning Meetings

This difference is such that the practice of breaking the product backlog items into sprint backlog items was unnecessary for the team since the items were already of an appropriate size for the sprint backlog.

The reason why these meetings were different from the ones in Scrum might be related to the differences in the Backlog as well as the fact that the Customer did not partake in the meetings. As our interview with the customer shows, it should be possible to get regular meetings with a representative. Another reason that I believe is more probable, is that since the team has no proper Scrum Master and understanding of the method, they don't really see a problem with the current situation. They get a prioritized list from the customer and then they're left in peace to work on this list. The group seemed to be quite happy with this solution.

They did mention that the customer wasn't as involved in the project as might have been preferred, but this might be because they know that they're supposed to have a highly involved user.

5.3 Possible Improvements

These suggestions are given in light of my observations and knowledge of development methods. So, while some of them should be quite good, others might be naive.

5.3.1 Training

It seems that none of the developers on the team received any education on the scrum method. Since such knowledge would equip the developers with ideas and proven solution to common problems, I believe that this could help the team improve their method further.

5.3.2 The Product Backlog

While I can't show any evidence for this, I suspect that a product backlog that contains features and change requests written by the customer is a better solution than the one employed by the team. This would provide both the team and the customer with a clearer impression of the project progress, and it would make it easier for the customer to prioritize the tasks.

Since the project has been operating with the current product backlog for over a year, and the customer doesn't seem to think of it as a problem, I think the team would do best if they did not change this during the rest of this project. However, I recommend that they look into using the scrum product backlog on future projects.

5.3.3 Meetings at the End of the Sprint

The team had a meeting at the end of the sprint which I called the sprint pre-planning meeting (Section 3.4.5). During this meeting the team accomplishes tasks found in both the sprint review meeting and the sprint retrospective meeting from scrum. The reason why Scrum divides this in two meetings, is that the customer only needs to participate at the first meeting. Separating these meetings make it easier to invite the customer since the meeting is shorter and more relevant to them.

To be able to have the customer attend regular meetings, the team need to make sure that the meetings are valuable to the customer. The first step in accomplishing this is to remove things from the meeting agenda that isn't relevant to them. The sprint review meeting seems to me to be something that can be inspiring for both the customer and the team, and so I recommend trying to move the items that belong in this meeting from the sprint pre-planning meeting and bring in the customer for a demonstration of what was accomplished during the sprint.

If this works and the customer manages to attend this meeting, it should be easier to have him attend the sprint planning meeting later.

Chapter 6

Conclusion

My first impression when I observed the major differences between the method employed by the team and what I knew was “right” after reading about XP and Scrum, was that this team didn’t really get it. They hadn’t read the books and they didn’t even seem embarrassed about it.

However, during my interview with Customer 1, this impression changed a bit. Yes, I still have a hard time understanding why they haven’t taken the time to read more about the method they are inspired by, but for some reason the customer seemed very happy with their work. To paraphrase him

> We just tell Developer 1 and the others that “this is the way we want it”, and that’s the way it ends up

and when I asked him to sum up and tell me how the cooperation between the customer and the developers was he said (again, I paraphrase)

> Fantastic! (. . .) They don’t know about the inspections and rules, and because of that they see the project in a different way than I do. This results in discussions leading up to very good solutions. A lot of good things have come out of the fact that both we and the developers have to think in new ways. I think this is very good.

Now if we look at the agile manifesto (see Section 2.4), it says

> . . . we have come to value: individuals and interactions over processes and tools

So, while I believe that the team can learn a lot from reading more about development methods, I have come to realize that the team knows how to develop software, and they are doing a good job at being flexible and adapting their method to suit their customer. Since the goal of the Scrum method is to make the customer happy, the team is already doing a fine job at it.

My interviews with the developers shows that they think the development method they are using is a step forward from their previous experiences. This leads me to believe that they will share their positive experiences with their friends and colleagues and be eager to use a similar method in the future. Maybe this is the reason why agile development methods are getting so popular? We all know that a happy customer makes for good PR.

Given the situation the team were in when they adopted Scrum, I think they did a fine job at it. Most likely they would have ended up with a different result if they were guided by a Scrum Master, and maybe this result would have been better, but my impression is that the team is effective and deliver what the customer wants. Since this is the goal of any development method I can't really justify criticising them.

Can they improve further? I suspect they will. They are continuously improving their method and learning from each other and from their coworkers.

6.1 Future Research

One of the questions that is left unanswered from this study, is what the consequences of the product backlog difference was (Section 5.1.3). It should be possible to do a controlled experiment that would answer this question. I suspect that the Scrum product backlog would make it easier to discuss the tasks with the customer, but to find out for sure, the question must be researched further.

6.2 What Have I Learned from this Project?

During these months of inspiration, writers block, discouragement, writing, encouragement and finally panic I have learned a lot about development methods, doing interpretive research, conducting interviews and thesis writing.

If I had the chance to do this thesis again, there are a quite a few things I would have done differently. The most pressing of these is the time devoted to the case study and the amount of interviews I did. I realize that one of the reasons why I had such a hard time writing on the thesis was that I didn't understand the case properly. Only after my interview with the customer did I realize this. Thanks to Developer 1, I was able to send emails with questions and have them answered the following day, something that I believe saved my project.

Looking at the quality of my first interviews and the quality of my last interview is also an eye opener. It seems that doing interviews takes practice and that I didn't have any.

Appendix A

The Research

In this chapter I give a summary of my visits to the team and how my interview with the customer came about.

A.1 The First Visit

During the first visit I spent much of the time getting familiar with the people and company culture of Company 2. I tried to learn what the company was expecting from me in return for letting me study them as well as getting an overview of the development method the team were using. My goal was to establish a connection so that I could come back later to do a more thorough study on the topics that I found interesting during this visit.

A.1.1 Day 1, Thursday

I arrived at Company 2 around 11:30 together with the researchers from SINTEF (see the preface). We were received by Leader 1 and Senior 2 who led us to their meeting room. Here we ate lunch and, while I stayed mostly in the background, the others discussed a research project they were engaged in as well as many other topics regarding agile software development and software development in general.

After this meeting the researchers from SINTEF left while I stayed and sat in on the company's bi-weekly staff meeting. This meeting wasn't very relevant for the study, but it helped me attain my goal of establishing a connection. In addition it gave me an impression of the company and employees, something I felt would make it easier to understand what I observed.

Finally there was a workshop for those interested in learning Ruby on Rails, a new technology that has become very popular recently. I joined in on this since I was curious on how this technology worked and also because I wanted the people in Company 2 to become familiar with having me around.

A.1.2 Day 2, Friday

I started out this day with a meeting with Leader 1 and Developer 6. Developer 5 was supposed to attend as well, but she had called in sick. During this meeting I got an in depth presentation of the project I was to study. We also discussed what my paper would cover and what Leader 1 was interested in learning from my study.

After this meeting Developer 6 spent some time answering some of my questions regarding the project and how the team worked. This session was especially useful for me since it became a very informal discussion of topics I felt that I had a good understanding of. It built my confidence quite a bit and made me feel more at home.

I ate lunch together with the other people at Company 2 and got to meet a few of the other developers that worked there. The discussion was not very work oriented this day since five or six of the people there were going on a skiing trip the following weekend and they were eagerly discussing this trip and trying to convince others to join in.

After lunch I did what might be called a very unstructured interview with Developer 6. I intended to do another semi-structured interview with him later when I knew more about what would be interesting for my paper, but since he wasn't available on my second visit this never happened.

A.1.3 Day 3, Monday

I observed Developer 6 work and prepare for the sprint planning meeting that we were to attend at Company 1 after lunch. This didn't take very long since Developer 5 was still sick. The rest of the time before the meeting he spent writing software while I read about the Scrum development method.

Senior 1 was going to participate in the meeting so he joined us when we left for Company 1 around 11. We arrived just in time for lunch and ate together with the team that were stationed there. The food was good and we had a very interesting conversation about development methods and Senior 1 told us about his experiences while working for Thoughtworks in London. Unfortunately I didn't take any notes of what was said. By now all the members of the team had met me and seemed to be comfortable with having me around as they acted friendly and made me feel very welcome.

After lunch we started the sprint planning meeting. There were five developers present and two observers; me and Senior 1. The reason why Senior 1 was present was to introduce the team to a new way of doing task estimation called planning poker. I've written about this in Section 3.4.8. I took notes but didn't say much except once or twice during a discussion about agile methods. It was clear that this planning meeting was different from the norm as they had Senior 1 present and much of the discussion was about different agile practices they could use to improve their method. They did however describe to us observers how these meetings usually went.

A.1.4 Day 4, Tuesday

I arrived at Company 1 at 10:00 to participate in the daily stand-up meeting. This was the first time I was able to observe one of these meetings.

The rest of the day I spent writing out my notes and trying to decide what I should focus on in the rest of my study. I didn't pay much attention on how the developers worked except to note that the two most senior developers were in one office and

the two newest developers were in an adjacent office.

During lunch Customer 1 joined us at our table and I got to talk to him a bit about his impression of the development project. He had only good things to say about the project and the team and seemed more annoyed at his own organisation for not devoting more resources to the project so that it could move faster. He said that there were people in the field who were eagerly awaiting the new product and that he was really looking forward to being able to give it to them.

A.2 Second visit

On my first visit I followed my councilors advice about continuously taking two sets of notes in a diary; what I observed and my own thoughts and feelings. On my second visit I was not as smart. I did take notes and I recorded the interviews I performed, but I did not take notes on most of the other events during this visit.

When I'm writing this chapter I realize that my councilors advice was a very good one since the notes I considered unimportant have been a great help for remembering what I did during the first visit. Luckily the second visit is quite recent and due to this I was able to reconstruct the days of it pretty well.

A.2.1 Day 1, Monday

I arrived at Company 1 at 10:00 to participate in the daily stand-up meeting. The meeting was quite surprising to me as I expected to see Developer 1, 2, 3 and 4. However, Developer 4 had been replaced with Developer 5 and Developer 2 was at the dentist and couldn't make it to the meeting.

After the meeting I learned that Developer 4 had been moved to another project very suddenly. The internationalisation task was complete and Developer 5 had been moved to the main team while Developer 6 was now working on some other project.

At the stand-up meeting Developer 5 reported an impediment that led to a design meeting straight after. I sat in on this meeting but didn't really learn anything interesting except for the fact that they did a good job getting the new developer up to speed and made her feel like part of the team. Developer 2 arrived during this meeting but decided to work on his tasks instead of joining in.

I ate lunch with the team, but we didn't discuss anything related to the project.

After lunch I participated in the sprint planning meeting. As mentioned above, they decided to try out a new estimation technique at the planning meeting I observed. This was the fourth meeting they did when using this method and so they were quite effective by now. I learned that they were only doing 50% of the tasks using this estimation technique however since they were helping out a study of whether this technique resulted in better estimates.

The meeting was done at 16:00 and that was the end of the day. I talked to Developer 1 about the possibility of doing a few interviews during the next few days. We ended up with scheduling interviews with Developer 1 and Developer 2 on Tuesday and the last two on Wednesday.

A.2.2 Day 2, Tuesday

After the stand-up meeting I made ready for my first interview. Developer 1 decided to go first and we started at 10:30. The interview lasted for 30 minutes and after a 5 minute break I did the interview with Developer 2. I found the second interview harder, but all in all I was quite satisfied with my interviews.

It was now time for lunch, so we went to eat at the cafeteria. As usual we all sat together and talked about different things. The topic was not very work related.

I spent the rest of the day transcribing the interviews and left work at around 16:30. On my way out I met Customer 1 again and I used the opportunity to ask whether he was available for an interview the following day. He said that he'd try to fit it in but that he couldn't promise anything. He told me that if we couldn't make it then I should contact him and we could arrange an interview some other

time.

A.2.3 Day 3, Wednesday

This was the final day of my second visit. My day started as usual with the stand-up meeting. After this I spent some time improving my interviews based on what I had learned from the first two interviews. I scheduled the days interviews for after lunch since I was trying to fit in the interview with the customer. Unfortunately he couldn't make it.

Lunch went as usual, todays topic was what you could use the extra long weekend for.

The interviews went pretty well, they were a bit shorter since I was more focused on what I wanted to know and also since Developer 3 and Developer 5 were relatively new compared to the others so I didn't ask them much about the history of the project.

I spent the rest of the day transcribing interviews, and left around 17:30. I left together with Developer 1 and we discussed the project, my research and the Scrum method. I talked to him about what thoughts I had formed so far and he gave me some feedback on this.

A.3 Interview with Customer 1

I contacted Customer 1 on the 29th of May to schedule an interview. He was busy but we made an appointment to get back to each other on the 1st of June. We finally managed to do an interview on the 7th of June. He picked me up on campus and drove me to a location his company had near by.

I got a lot of interesting information from this interview to back up and tear down some of my theories. We spent about 30 minutes on the interview, after which he drove me back to campus. During the drive we talked about the project and I wished him luck with it and thanked him profusely for his time.

Appendix B

Interview Guides

Since the case I studied was a project that had been running for some time before I entered the scene, there was a lot of interesting events and decisions I didn't have the opportunity to observe. To form an idea of what these events and decisions were and how they were handled, I interviewed some of the people involved. For Developer 1-3 and 5 I used the first interview guide. For the customer I used the next one. I did not use an interview guide in my interview with Developer 6.

B.1 Developer Interview Guide

- Når begynte du på dette prosjektet og i hvilken rolle?
  - Mer om prosjektet da: Utviklingsmetodikk, besetning, kundeforhold, ...
  - Hadde du noen erfaring i bruk av denne utviklingsmetodikken?
  - Hadde du noen erfaring i bruk av Scrum eller andre smidige metoder?
- Hvordan opplevde du overgangen til å bruke Scrum?
  - Var du delaktig i bestemmelsen?
  - Kom det overraskende på deg?

  - Fikk du noen kurs eller annen opplæring i metodikken?
  - Var du positivt innstilt til denne måten å jobbe på?
- Hva var førsteinntrykket ditt av Scrum?
  - Hvordan har dette inntrykket endret seg?
- Hva synes du er positivt med Scrum?
- Hva synes du er negativt med Scrum?
- Hvordan er kommunikasjonen innad i teamet fungerer?
  - Kan Scrum ha påvirket måten dere kommuniserer på? Positivt eller negativt?
- Hvordan er kommunikasjonen utad mot kunden?
  - Har Scrum noen innvirkning her?

B.2 Customer Interview Guide

- Hvordan ble kunden innvolvert?
- Hvilken rolle har kunden i forbindelse med prosjektet?
- Hvilken autoritet har kunden?
- Hvem bestemmer hva som er viktig og som skal bli gjort i forbindelse med produktet?
- Hvordan går kunden frem for å få gjort endringer i produktet?
- Hvordan vet kunden hva som blir utviklet?
- Hvilke erfaringer har kunden fra tidligere utviklingsprosjekter?
- Er ikke med på planleggingsmøte, hvorfor?
- Kunne du vært med på et fast møte med utviklingsteamet annen hver uke? Hver måned?

- Forhold til Scrum?
- Ansvarsforhold?
- Rolle
- Involvering
- Erfaringer
  - Tidligere
  - Prosjektet
  - Metodikk
- Resultat
- Problemer

Bibliography

[All01] Agile Alliance. Manifesto for agile software development, 2001. [Online; accessed 17-December-2005].

[Bec99] Kent Beck. Embracing change with extreme programming. *Computer*, 1999.

[Bec00] Kent Beck. *eXtreme Programming Explained*. Addison-Wesley, 2000.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[Boe88] Barry Boehm. A spiral model of software development and enhancement. In *IEEE Computer*, volume 21, pages 61 – 72. IEEE, May 1988.

[CH04] WenShin Chen and Rudy Hirschheim. A paradigmatic and methodological examination of information systems research from 1991 to 2001. *Information Systems Journal*, 14:197 – 235, 2004.

[Coc00] Alistair Cockburn. Reexamining the cost of change curve, 2000. [Online; accessed 18-December-2005].

[Coc02] Alistair Cockburn. *Agile Software Development*. Addison-Wesley, 2002.

[Coh05] Mike Cohn. *Agile Estimation and Planning*. Prentice Hall, 2005.

[Con06] Conchango. Scrum faq. *Web*, 2006. [Online; accessed 18-April-2006].

[Fow00] Martin Fowler. *Refactoring*. Addison-Wesley, 2000.

[Fow01] Martin Fowler. Is design dead? In *Extreme programming examined*, pages 3 – 17. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[FPB78] Jr. Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.

[Gal94] Robert D. Galliers. Choosing information systems research approaches. In *Information System Research. Issues, methods and practical guidelines*, pages 144 – 162. Blackwell scientific publications, 1994.

[Gil76] Tom Gilb. *Software Metrics*. Little, Brown, and Co., 1976.

[GL87] Robert D. Galliers and Frank F. Land. Choosing appropriate information systems research methodologies. *ACM*, 30(11), 1987.

[Gre02] James Grenning. Planning poker. *Web*, 2002.

[Jar99] Stanley J. Jarzombek. Proceedings of the fifth annual joint aerospace weapons systems support, sensors and simulation symposium. *Government, Printing Office Press*, 1999.

[Joh99] Jim Johnson. Turning chaos into success. *Web*, 1999.

[KM99] Heinz K. Klein and Michael D. Meyers. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 1999.

[LB03] Craig Larman and Victor R. Basili. Iterative and incremental development: A brief history. *IEEE June 2003*, 2003.

[LC02] Theron R. Leishman and Dr. David A. Cook. Requirements risks can drown software projects. *CrossTalk April 2002*, 2002.

[McC04] Steve McConnell. *Code Complete*. Microsoft Press, 2 edition, 2004.

[MJ82] Daniel D. McCracken and Michael A. Jackson. Life cycle concept considered harmful. *SIGSOFT Softw. Eng. Notes*, 7(2):29 – 32, 1982.

[MS02] Kane Mar and Ken Schwaber. Scrum with xp. *Web*, 2002.

[Ree04] Dafydd Rees. Agile project management with scrum. *Web*, 2004.

[Roy70] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. In *Technical Papers of Western Electronic Show and Convention (WesCon)*, 1970.

[Sch03] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2003.

[vV00] Hans van Vliet. *Software engineering (2nd ed.): principles and practice*. John Wiley & Sons, Inc., New York, NY, USA, 2000.