IT 09 024
Examensarbete 30 hp
Juni 2009

Requirements Engineering in an Agile Environment

Yunyun Zhu

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Requirements Engineering in an Agile Environment

Yunyun Zhu

The Requirements Engineering (RE) process often dominates the quality of a project. The requirement practices in a project team are supposed to be an important part of the whole software development process. Today lean and agile development is becoming more and more popular in industry. Many project teams work in an agile environment in order to have rapid delivery of high-quality software. Usually the work of the teams adopting an agile methodology is flexible and changeable. This indicates that the requirements of the projects might also be frequently changed, which is a variation to the traditional RE that relies on long and detailed documentation.

This thesis investigates how to conduct a RE process under an agile environment – so that there exist relatively formal but agile and changeable requirements within a project. The methods planned to be used are literature study, a case study carried out in two software development teams in the Test Tool & Support Section at Sony Ericsson Mobile Communications AB, and one pilot in each team based on the case study. There were 11 employees interviewed, including eight developers, two product owners and one scrum master. The evaluation on the pilots was mainly based on the feedback from the interviewees on the pilot.

The outcome of the evaluation was that one of the teams (BRAT team) should adopt user stories for user-related requirements, "done criteria" and non-functional requirements, and have the product owner to do the demonstration during the sprint review in the future. Also, when budget allows, they should have one or more full-time testers in the team and formal documentation of the requirements. Besides the suggestion for the BRAT team, QC team was suggested to have a glossary, formalize the defect description and have the product owner to ask the customers for the feedbacks on the developers' thoughts about the uncertain requirements.

Handledare: Anders Nyberg
Ämnesgranskare: Roland Bol
Examinator: Anders Jansson
IT 09 024
Tryckt av: Reprocentralen ITC

Acknowledgements

First of all I would like to thank my project supervisor Anders Nyberg at Sony Ericsson, and my examiner Roland Bol at Uppsala University. Thank you for many constructive suggestions and instructions. The thesis work would have been so much harder without your help.

Thanks to the scrum master, the product owners and all the developers in both the BRAT and the QC teams in the Test Tools & Support Section for participating in my interviews. A special thanks to the BRAT team, for taking part in the pilot. I could hardly draw the conclusions I did without the pilot sprint and your feedbacks.

Also, I like to thank Ivan Christoff, for nice and kind encouragement during my thesis period in Lund.

Finally, many thanks to my loving parents, who have supported my master studies in Sweden both mentally and materially.

Thank you all!

Uppsala, June 2009
Yunyun Zhu

Table of Contents

1 Introduction

1.1 Background

Requirements Engineering (RE) is a principal area in software development and production. A high quality RE process often dominates a successful project [11]. Traditional Software Engineering (SE) usually includes the RE process which consists of requirements elicitation, analysis, documentation, validation and management.

In recent years, many industrial project teams work in an agile environment in order to have rapid delivery of high-quality software. Different from traditional SE process, agile method relies on iterative development and face-to-face communication. Among all the agile developments Scrum is one of the most popular methods in industry [2]. It separates a project into sprints, with each sprint being two to four weeks long, and has new delivery after each sprint. Tasks to be done are discussed before every new sprint starts in order to meet customers' changing needs. The requirements in Scrum development will evolve rapidly during the project in this way.

1.2 Problem description

The test tools section at Sony Ericsson Mobile Communications AB (SEMC) is working hard to become a more agile and lean software development organization. The teams are working according to Scrum, using test driven development, continuous integration etc. The work is well along this track, but the teams lack a good, lean and agile way to collect requirements from the rest of the organization and a way to investigate the impacts when changing the requirements. In this way, they need a more formal Requirements Engineering process.

However, traditional RE, which requires long meetings and documents, is usually pre-designed step by step preciously. This process is obviously inapplicable to the quick and changeable work under an agile environment. Thus, the problem that needs to be solved is how to conduct a RE process in a scrum environment - so that there will be more formal requirements at the beginning of the project but the requirements can still be changed and the changes are traceable.

1.3 Purposes

This thesis project aims to

i) investigate the working process of the two teams in the test tools section at SEMC
ii) find out the problems the teams have in working with the current requirements
iii) design some possible improvements based on the discovered problems
iv) try out the improvements in one pilot for each team
v) evaluate the effects of the pilots in order to propose a suggestion on the future work of the teams

1.4 Delimitations

The goal of this thesis project is to find out an effective way to conduct a Requirements Engineering process in an agile environment. The newly-found methodology will only be tried out within some small pilots and the tools used in the teams will not be evaluated.

The testing work in the scrum team will not be investigated deeply, but there were another two students who were working on this topic in the Test Tools & Support Section meanwhile [16]. So sometimes there can be some interaction between this thesis and the testing thesis.

2 Method

This section describes and motivates the methods applied in this study. There are four parts in the section, including *theoretical viewpoint*, *design of the study*, *research quality* and *practical process*.

2.1 Design of the study

The thesis project is carried out according to Figure 2.1.

When the thesis began I needed to gather relevant theories and this started with literature studying. The collection narrowed down after a period of searching for general knowledge from books, academic papers and professional magazines.

During the theory learning time it was also necessary to get familiar with the teams' working process and later identify the problems they have. Since the goal of the thesis project is to improve the RE process within these two teams, I chose to use case study in order to investigate how the teams worked with the requirements in an agile environment.

A case can be an individual, an institution or a large-scale community, so a case study is one which investigates a case to answer specific research questions, which seeks evidence and which is collated to get the best possible answers to the research questions [12]. Case study is also the study of the particularity and complexity of a single case, coming to understand its activity within important circumstances [34]. So it was applicable to adopt case studies within the teams.

In order to have an overview about how the team members looked at the requirements process they are working with, I chose to conduct qualitative interviews within the teams. Qualitative interviews aim to be more experienced at the phenomena and to deeply understand the issues. It is consistent with the objective of the thesis of having a thorough and deeper understanding of the current situation of the teams and identifying the problems they have.

The techniques used to collect theories are described below. The parts of analysis, problems identifying, solution and evaluation will be discussed in the later sections.

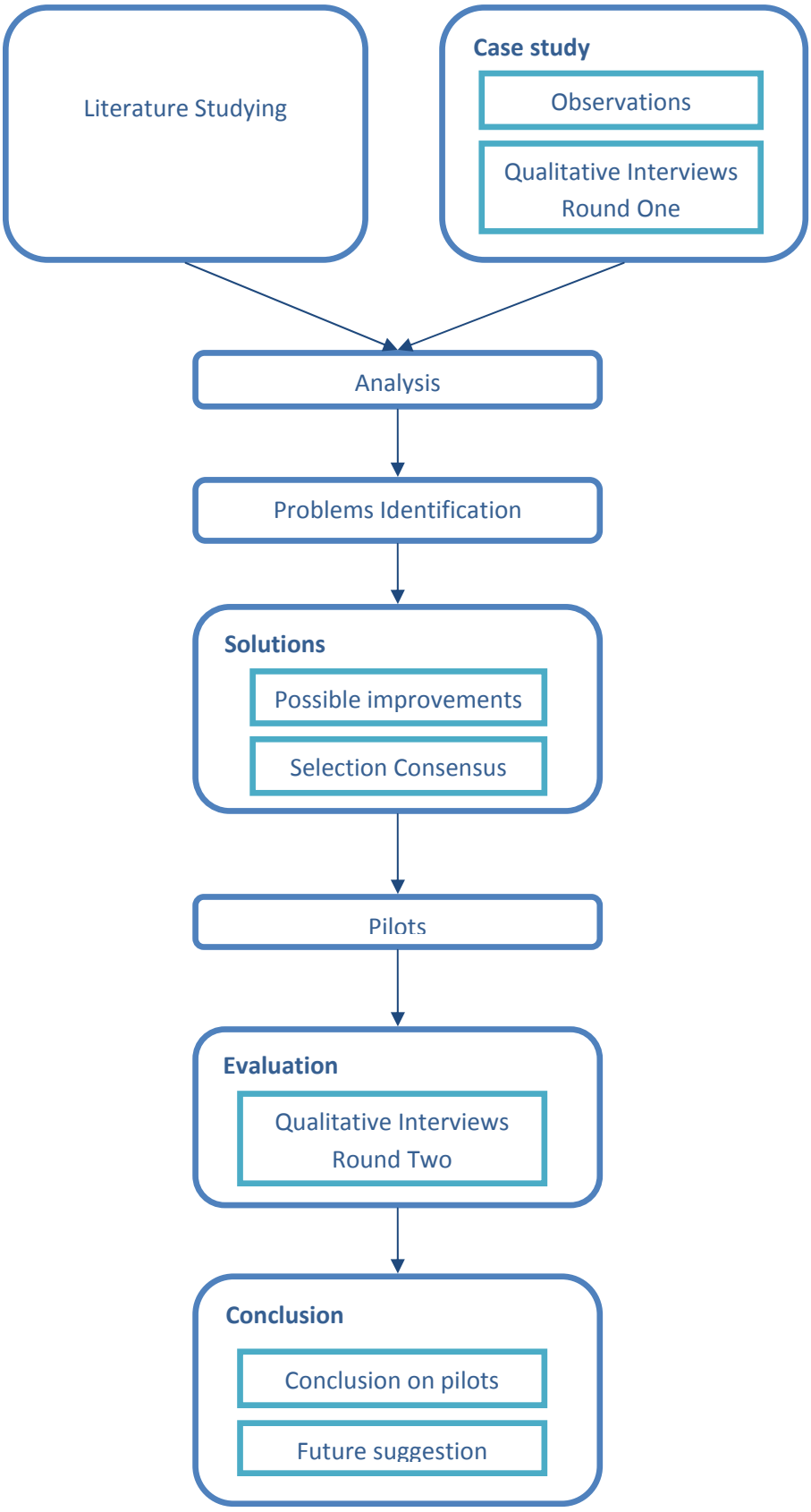
Literature Studying
Case study
Observations
Qualitative Interviews Round One
Analysis
Problems Identification
Solutions
Possible improvements
Selection Consensus
Pilots
Evaluation
Qualitative Interviews Round Two
Conclusion
Conclusion on pilots
Future suggestion

Figure 2.1

2.1.1 Literature study

This thesis covers the areas of both RE and agile software development, so I decided to develop my knowledge firstly in RE and Agile development, especially the Scrum method. For the agile development, I especially explored the topic of Requirements Engineering and software testing in an agile environment.

Since both agile development and requirements engineering derive from practical work, the literatures should include instruction books about how to carry out a project in industry as well as articles from academic field. Below are the areas I chose to do the literature studying in:

- **Requirements engineering:** the concepts of Requirements Engineering, and the practices that should be conducted in a formal RE process. The intent is to analyze the necessary practices in a traditional RE process and find out which of them can be used in an agile environment.

- **Agile software development:** the concepts of agile software development. The intent is to identify the characteristics of Agile by comparing with the traditional software engineering, and introduce how RE and software testing are conducted in an agile environment.

    - **Requirements engineering in agile development:** the instructions and examples of RE processed in agile development. The intent is to compare the examples with current situation and see if there is any practice that can be adopted by this project (after some modification).

    - **Software testing in agile development:** the process of software testing conducted in an agile environment. Software testing within the scope of this thesis, but it is related to the requirements of a project. The aim of looking into this topic is to find out if there is any testing methods that can be helpful to the RE, e.g., to help with the requirements validation.

- **Scrum:** the concepts of scrum, one of the common methods of agile development. The intent is also to identify the characteristics of scrum and to figure out what practices of traditional RE can be suitable for a scrum team or what practices need to be modified to match the iterative development.

2.1.2 Interviews

As discussed earlier, qualitative interviews were performed within the two teams in Test Tool & Support Section of Sony Ericsson. The reasons for this were

i) The teams worked according to scrum and thus there was not enough documentation which described the work process and all relevant elements. I needed to talk to the team members to have a complete overview of the teams' work.

ii) The scrum work was conducted by people. It was hard to understand how the work was carried out only by reading theoretical literatures. To know about the teams'

practical feelings and feedback was also necessary since not all the theories can be applicable to real world.

The objects of interviews included all the roles in the scrum work. They were:

- 1 scrum master
- 2 product owners
- 8 developers

I first interviewed the scrum master in order to have a general idea about the whole working process of the teams and the responsibilities of different roles. Then I interviewed the product owners and the developers to collect their feedback on the current work situation. As it is showed in Figure 2.1, this was the first round of the interviews. After the "possible improvements" were tried out in the pilot. The second round of interviews was conducted to evaluate these improvements.

Different questions were designed according to the roles of the interviewees, but all the interviews were loosely-structured. I prepared in advance some main questions and open questions, which were meant to lead the topics. Meanwhile the interviewees were free to discuss whatever they thought which was relevant to the interview topics.

2.1.3 Observations

Besides interviews, observation was also a resource from which I got information about how the work was carried out in the teams. I attended almost all the meetings that were related to requirements (for more details about the functions of the meetings see section 3.3.1):

- Daily standup meetings, which were held every morning during the sprints and lasted for about 15 minutes each time.
- Poker planning for the next sprint, which was held in the middle of the current sprint and lasted for 2 hours.
- Sprint planning, held at the beginning of each sprint and lasted for 4 hours.
- Sprint retrospective, held at the end of every 2 sprints and lasted for 1 hour.
- Sprint review, held at the end of every 2 sprints and lasted for 1 hour.

During the meetings, I listened to the discussed contents, observed the interaction among the team members, their behaviours, etc. Also I tried to find out the problems that the teams had and the places that could be improved. All the findings and thoughts were noted down in the meetings.

Furthermore, I had a desk at SEMC and sat with the team members every day. I could notice all the activities in the teams, and this helped me to know the current situation of the teams' work better.

2.2 Research Quality

This thesis was mainly based on case study. The Case study method has the limitations [28] such as not being generalizable enough because it is impossible to generalize from a sample [33]. Since the goal of the thesis project was to conduct Requirements Engineering only within the two teams in Test Tools & Support Section at SEMC, the research would not be influenced because of the case study being focused on limited number of people. Also different methods such as interviews and observations were performed, this could reduce the amount of missing important information.

Before the pilot, I performed every interview individually, face-to-face and guaranteed to the interviewees that their answers would be only used for the thesis. In this way the interviewees could feel free to discuss any problems they thought about. Furthermore, in order not to let the speaker feel any pressure, I did not use any audio recorders. The answers were noted down by hand and later recorded into Microsoft Word right after every interview.

2.3 Time Frame

The project was started in the middle of December in 2008. I was mostly doing the literature studying and collecting the relevant information from the Internet until the middle of January, 2009, when I moved to SEMC and started the case study on the two teams at Test Tools & Support Section.

During the first two weeks at SEMC, I investigated the working process of the two teams by observations and talking to the teams informally. In the following five to six weeks I attended all the meetings with requirements involved, including poker planning, sprint planning, sprint review and retrospective etc, and interviewed the product owners of both teams.

In the last two weeks of March, I interviewed most of the developers of the two teams and figured out some possible improvements which were intended to solve the problems I found from the meetings and the interviews.

At the beginning of April, I held two meetings with both teams. We discussed the possible improvements and the teams selected the ones they would like to try. BRAT team tried out a pilot sprint at the beginning of May as planned, and I collected the feedbacks on the pilot from the developers and the product owner on May 19th and 20th. The presentation of the project thesis was done on May 29th at SEMC, the last day of my thesis work there.

3 Theoretical Framework

This section has an overview of the theories which are relevant to the Master Thesis. The first part introduces Requirements Engineering. Non-functional requirements and user stories, which are relevant to the teams' work in practical, are specifically mentioned. The next is agile development part. Besides the concepts of agile, RE and testing in agile environments are also described. Finally is the introduction of scrum, the method that the teams in Test Tools & Support Section are currently working with.

3.1 Requirements Engineering

Generally speaking, software requirements refer to the description of the purpose that a system is intended to. They can be regarded "as a specification of what should be implemented" [20]. Requirements indicate what customers really want, the functionalities that a system is supposed to provide to satisfy the customers, the constraints of the system etc. It is important to gather complete requirements so that developers will clearly know users' needs and thus decide how to implement the system.

Requirements engineering is the process of identifying relevant users and analyzing their needs to find out the intent of a system. It is a systematical series of activities to be conducted on requirements. Today there are many different definitions of Requirements Engineering. One of the most precise descriptions, in my opinion, is Zave's [41]:

"Requirements engineering is defined as the branch of Software Engineering concerned with the real-world goals for, functions of, and constraints on software systems; it is also concerned with the relationship of these factors to precise specifications of software behaviour and to their evolution over time and across software families."

Requirements engineering contains the activities of

- Requirements elicitation
- Requirements analysis
- Requirements documentation
- Requirements validation
- Requirements management

One thing worth being mentioned is that requirements validation and requirements management are as necessary as the other three requirements practices in the RE process.

The object of validation is to make sure that there is nothing missing from the requirements, there is no contradiction against the standard and all the requirements are clearly described. The validation should take place when all the requirements are included in the final draft. It is hard to find something specific to follow while validating the requirements, but it is a chance for people to ensure that they will have the requirements which represent the customers' real needs before they start the development phase, since finding out and mending the problems in a requirements document will help to avoid some rework later during the system implementation [20].

The goal of requirements management is to keep the requirements quality whenever there are changed or newly added requirements during the system implementation. The dependencies and the relationships of different requirements documents therefore must be managed, and all the requirements should also be ensured traceable, which helps to investigate the impact of the changes [20].

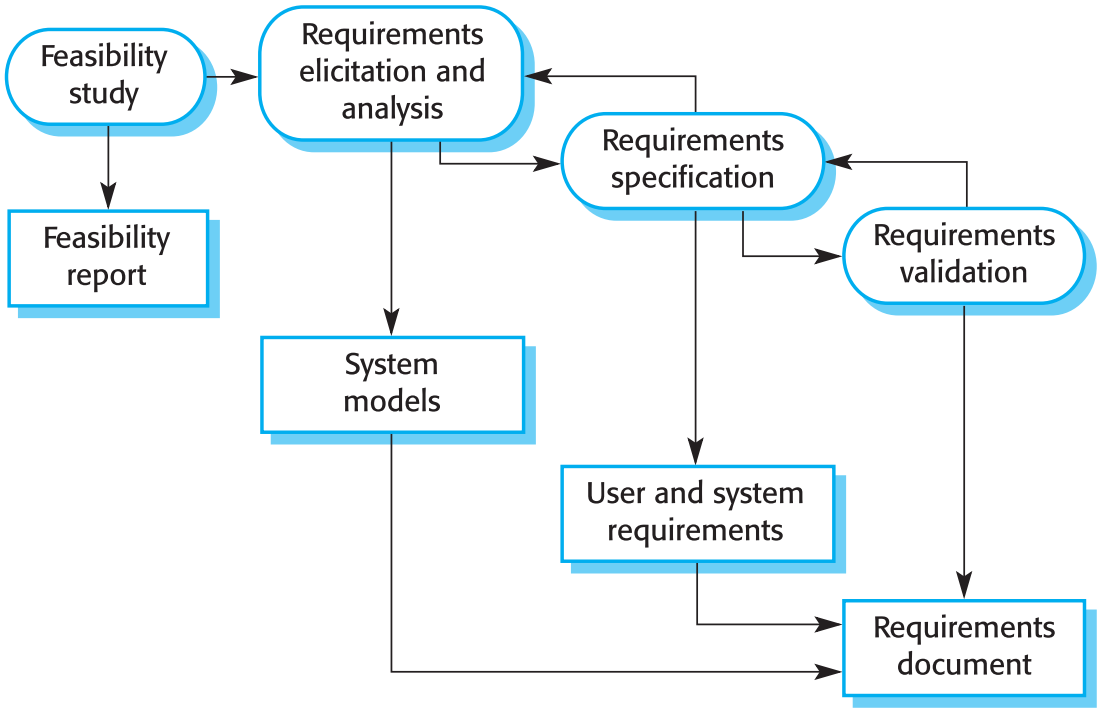

Figure 3.1: The requirements engineering process [21]

3.1.1 Functional requirements and non-functional requirements (NFRs)

System requirements are usually separated into two types: functional requirements and non-functional requirements [4].

Functional requirements describe what a system should do while non-functional requirements are constraints on the services or functions offered by the system [21].

Functional requirements are dependent of the end users and what kind of software that is developed, etc. [21]. They are related to the system's actions and can be regarded as business requirements, i.e., a user or a customer will be able to describe what the system is intended to do [15].

Non-functional requirements are seldom associated with single system features [21]. They often refer to the properties that the system should have and the way in which the customers want the product to act. (E.g., performance, security, usability etc.) [21] Also, some non-functional requirements may limit how the system should be developed [21].

NFRs sometimes are more critical than a single functional feature. Users probably can find a way to solve their problems without a function that specifically satisfies their need, but the whole system might not be able to work if some NFR is not met [21]. For example, an online

system which should be run by 100,000 users at the same time can only support 1,000 (say) concurrent users actually. In this way it will be collapsed soon when more users log in and start using the system. Also, when more than one product provides similar functions, the non-functional properties will be judged by users. The ones that are more convenient to be used or have higher performances will attract more customers [15].

There are three types of NFRs: [21]

- **Product requirements,** which specify product behavior. E.g., performance requirements or reliability requirements.
- **Organisational requirements,** which are from the customer's and the developer's organisation. E.g., the limitation of the programming language to be used.
- **External requirements,** which cover all requirements that are from factors outside of the system. E.g., the legislative requirements which mean that the system must act follow the law.

Figure 3.2 shows all the types of non-functional requirements defined by Sommerville [21].

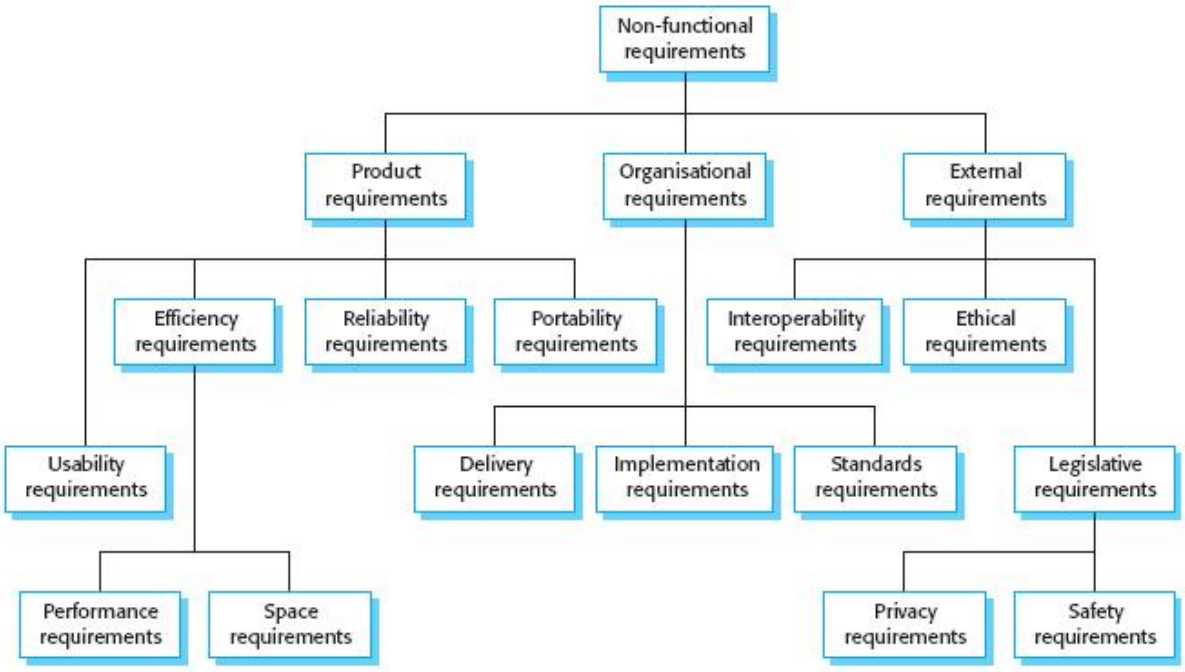

Figure 3.2 Types of non-functional requirements

One thing should be noted here is that it is always important to describe non-functional requirement quantitatively so that they are testable. However, sometimes it is hard to quantitate all the NFRs. Also, Sommerville suggests the metrics to specify them (Figure 3.3):

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

Figure 3.3 Metrics for specifying non-functional requirements

3.1.2 Use cases VS. User stories

The description of a requirement should be understandable, clear, concise and consistent. There are many ways to describe requirements. Here I would like to just introduce use cases and user stories, which are relevant to the case study of my Master Thesis.

Use cases are also called scenarios [20]. It indicates the size and shape of the target [18]. A use case describes the flow of a scenario, including:

i) the state of the system before the scenario happens
ii) a series of events happened during the scenario
iii) the expected result of these event
iv) what else might happen concurrently
v) the state of the system after the scenario

Requirements engineers go through the scenarios together with the end-users and generate the use cases later. Usually a use case will take three to four pages [20]. The aim of using a use case is to describe how a system, which is regarded as a black box, interacts with the world outside, including the users and other systems [18].

User stories are used for agile software development [5]. They describe very thin slices of work to be implemented [18]. A user story is usually just one or two sentences of description. It is written in the format of [31]:

- As a [**person in a role**] I want to [**perform some activity**] so that [**some goal is achieved**].

The aim of using a user story is to mark the requests for system functionality. It is just a marker rather than a requirements document. Later there will be more expansion on it when the relevant requirement is to be handled [18].

There are both advantages and disadvantages of using use cases and user stories.

A use case has the merits of [18]:

- Presenting the system behaviour clearly
- Being used to get a good handle on the size of the system to be developed and to decrease the unexpected condition happened
- Recording the results of the discussions
- Having a big picture to show all behaviour around a system request to make it easier for people to see what they ask for.

The limitation of use cases are [18]:

- Hard to implement the whole use case in a single iteration
- Not appropriate to be used in bug lists and feature enhancement lists, since a use case does not provide a real purpose there.

A user story has these benefits [18]:

- Short
- Smaller and thus easier to be estimated, prioritized and managed [3].
- Can be split as small as one wants, which means it can fit into any size of iteration.

One thing worth being noted is that Mike Cohn suggested that non-functional requirements can also be described in the format of user stories, and the advantage of doing it is that people will not forget the reason of having this NFR after sometime, especially when the NFR is an organisational decision [38].

Because user stories are extremely short descriptions of some single functionality to be implemented [11], one of the main limitations of writing user stories is:

- It is hard to cover large projects only with user stories

From above we can see that there are both pros and cons in using use cases and user stories. It is difficult to tell which method is better. In many cases people should analyse the practical situation in their development environment and then decide whether they should adopt use cases or user stories.

3.2 Agile Software Development

3.2.1 Introduction

In 2001, 17 people created Agile Software Development Manifesto in Utah in USA, which are the representatives from XP, Scrum, etc and an alternative to documentation-driven or "disciplined" processes [27].

Agile methods do not fix all the plans at the beginning of the project. Instead, the project is broken into smaller sub-tasks and they are implemented in short time-boxed iterations, with the goal producing shippable code incrementally [14]. All iterations have the same pattern, which has three phases. The first phase is a planning session, including the prioritization and the estimation of the working tasks and the team commits to the work. The second phase is the development phase which includes implementation and tests, and the last phase is the delivery of the increment. [14]. In this way, requirements may be introduced, modified or removed in successive iterations, and the agile development methods have the feature of embracing changing requirements during the process [32]. So the risk of changing the whole requirements, which is usually defined at the beginning of the project in a waterfall process (see section 3.2.2 for more information about the difference between agile methods and waterfall methods), can be reduced.

In agile methods, intensive communication and collaboration among the team members plays a crucial role. The size of the team is usually small, i.e., 5 – 8 persons per team. The team should sit in one room so that everyone is aware of others' availability to answer questions [9]. If all the members are local, the team should sit together to make the face-to-face talk easily. If the team is working in different places, they should overcome the situation by contacting via e-mails, instant messages, etc [9].

Every agile team should have a usage expert available all time, so that the developers will get quick feedbacks of the questions to users' needs and habits. The usage expert being on-site can help the development team make fewer mistakes [9]. At the end of every iteration, the team will have a review with the customers and demo what they have implemented in the past iteration, in order to get feedback and guidance from the customers [14].

Most agile methods, such as scrum, which will be introduced in section 3.3, follow a daily stand-up meeting among the team. Every day, the team members report what they did the last day, what they plan to do today and the issues they have met. In this way the progress made by everybody is shown of the current iteration and the problems are reported in time [14].

3.2.2 Agile development process VS. Waterfall process

Different from agile method, the waterfall model is more classical since it was derived from more general system engineering process in 1970 [21].

As the name suggests, waterfall model is a sequential manner. See Figure 3.4, the principal phases are cascaded from one to another. The former phases should be frozen when the work is continued to later stages [21].

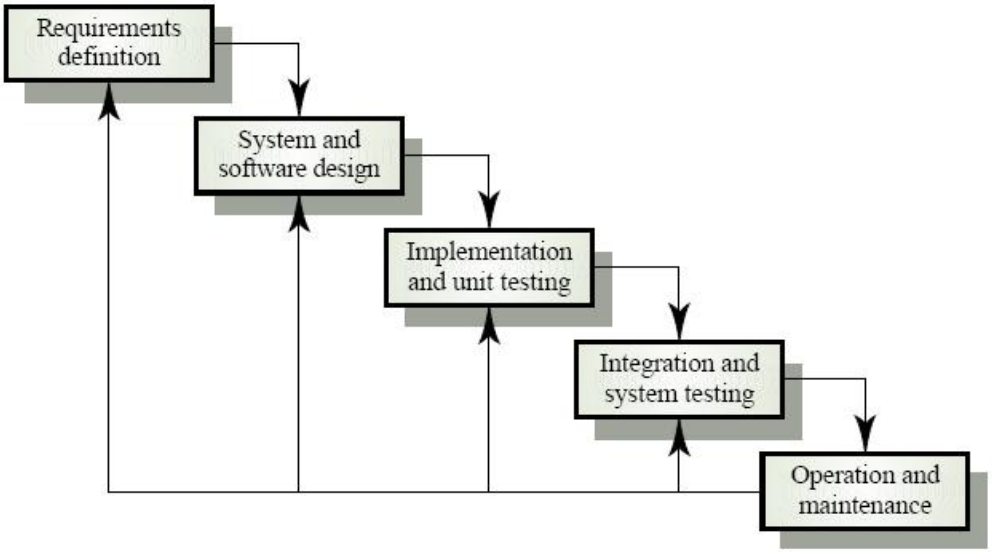

Figure 3.4 The requirements engineering process [21]

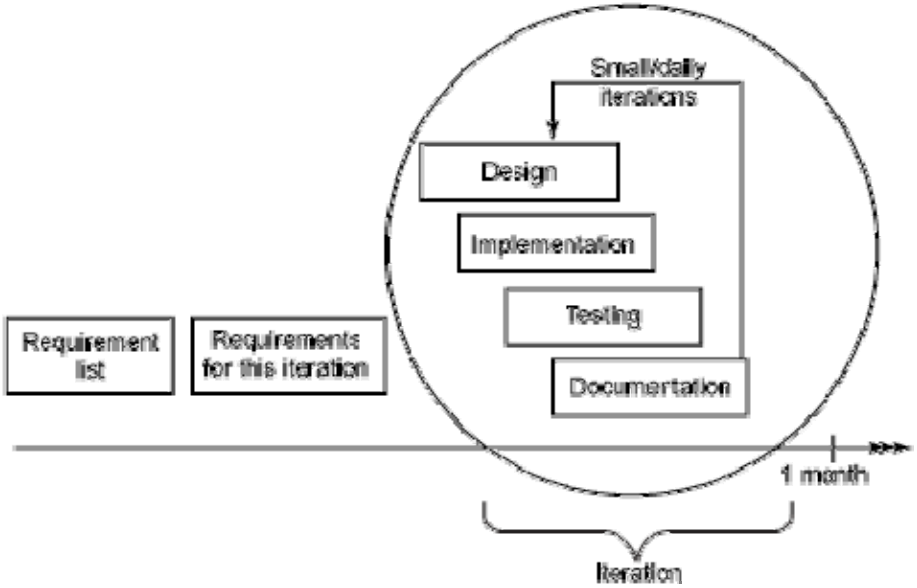

Figure 3.5 First iteration in an agile development process [8]

We can see that compared with agile method, the main problem of the waterfall method is that the project is separated into stages distinctly and commitments must be made at an early stage [21], which makes it hard to alter the requirements if customers change their minds. So waterfall is more suitable when the requirements will probably not be changed during the implementation time.

On the other hand, however, agile methods have short iterations which last for only a few weeks. As I have introduced in section 3.2.1, after each iteration, the requirements can be changed or some new requirements can be added into the projects. Moreover, if a project following waterfall model is canceled halfway, nothing can be shown to customers. In contrast, if the project is under agile method and is cancelled at any point, there is always something, that has been released in former iterations, to be displayed to customers [40].

3.2.3 Requirements Engineering in Agile environments

Among the researches concerning about requirements engineering in agile, there are four main focuses:

- **General description of RE practices in agile:**

Bose, Kurhekar and Ghoshal (2008) [39] highlight the requirements engineering modified to be adopted in agile methodology. They suggest that requirements should be gathered from different viewpoints of stakeholders and interviews should be used. The verification of early requirements descriptions and non-functional requirements being included are also proposed. Furthermore, they mention that the requirements management should be adopted to ensure the traceability, which is important when the requirements are likely to be changed in an agile method. Their conclusion is that agile requirements are effective to get continuous feedback from customers while the limits of agile methods are still not defined well.

Silliti and Succi (2005) [11] describe the agile approaches to requirements elicitation and management. The practices are focused on continuous interaction with customers to add new requirements and remove the out-of-date ones. The authors believe that the requirements should be collected by the whole development team discussing with the customers in elicitation phase, and the interaction between the developers and the customers should be direct and without any intermediaries. The requirements can be evolved by the collection and prioritization activities at the beginning of each iteration. They come to the conclusion that agile methods manage requirements effectively in small teams rather than big ones, and emphasize the importance of customer involvement so that the waste in requirements during the production can be reduced.

- **Comparing traditional requirements engineering with agile methods and trying to fit the RE process under an agile environment.**

For example, Eberlein and Leite (2002) [10] think that the requirements are poorly tackled in agile teams, and list out four RE practices - customer interaction, requirements verification & validation, non-functional requirements and change management - which they believe good for agile approaches but can be ignored by people in an agile environment. They think that customer interaction has been focused a lot in agile methods, but different interview techniques, especially the context-free questions can also be learnt. Requirements verification, according to them, can improve the quality of agile process if it is applied together with validation and some simple tools are also used to check the early requirements description. They also mention the need to include techniques of identifying non-functional requirements, and agile methods must have requirements management since it is the precondition to managing change of customers' needs.

Paetsch, Eberlein and Maurer (2003) [17] compare the differences and similarities between traditional RE approaches and agile methods and try to find what benefits of RE methods can bring to agile software development. They find that traditional RE does not have a good way of stakeholder involvement as agile methods. Also, the customers are

only involved in early stages in RE while they are involved during the whole process under agile. They think that agile methods lack enough documentation while traditional RE tends to over-document. Agile methods use throw-away models but RE has different levels of abstraction models. Furthermore, changes being traceable are regarded important in RE, and agile methods also manage the requirements well with index cards. Non-functional requirements are poorly defined in agile approaches according to them. Their conclusion is that the RE phases is not very clear in agile - they are merged and repeated in each iteration. Also, documentation is the main difference between the RE process and agile methods. In this way a minimum of documentation should be ensured in an agile environment and there should also be enough documentation in a team for future maintenances.

- **Description of the challenges and the advantages of agile requirements engineering.**

Tomayko (2002) [23] observes 12 XP (eXtreme Programming) teams and finds out that a rapidly developed prototype in agile methods usually answers a question about requirements content. Customers thus will know what else they want and get rid of the duty of providing correct requirements in advance. Another benefit of agile methods when requirements are unstable is that more defects detected by test-first programming in XP, and RE is advanced when the requirements can be tested. But he also illustrates that the incorrect estimation and the increased cost due to the altered architectures are the difficulties of requirements collection with agile methods.

Cao and Ramesh (2008) [26] investigate 16 software development organizations and analyse the pros and the cons of seven agile RE practices: face-to-face communication over written specifications, iterative requirements engineering, requirements prioritization goes extreme, managing requirements change through constant planning, prototyping, test-driven development and use review meetings and acceptance tests. They analyse both of the benefits and challenges of these practices and suggest that they are neither panacea nor poison to organizations and they should be used depends on the practical project situation.

- **Introduction of the techniques of RE in agile environments.**

Boness and Harrison (2007) [24] introduce a goal-sketching technique. They at first create a goal graph which is vague and about the current understanding of the overall intention. Then there come a series of stages of development, which is similar to scrum. Each stage takes a part of the graph, which is called stage graph, and refines it so that the intention of this part is clear. During a stage the stage graph might be updated and at the end of the stage, the system graph is updated with the completed stage graph. In this way, the goal graph emerges by the iterative developments. The technique has been tested successfully in some industrial projects.

Svensson, Birgisson, Hedin and Regnell (2008) [35] present a four-abstraction-level agile model, Agile Requirements Abstraction Model (ARAM), which is an adaption of Requirements Abstraction Model (RAM) used for traditional RE, integrated into a scrum

environment. The four levels are from general to specific: product level, feature level, function level and component level. Also, there are three steps to follow when the model is used in RE phase: elicitation step where original requirements are collected, placement step where each requirement is identified which abstraction level it is at, and work-up step where all the requirements, including the existing ones and the new ones, are linked together with ID numbers according to their "algorithm", so that the requirements can be compared during prioritization and are suitable for interring the backlog. If the customers want to follow a traditional path of development, one more step of documentation is added. The agile methods are only adopted within the teams so the customers only see the released results as milestones and don't have to be involved in the RE process. The evaluation shows that the model adopted in a scrum team can stay agile.

I tried to find out some information concerning about the Requirements Engineering in agile requirements, especially how RE process was executed in scrums. However, most of the available academic papers I found were about RE in general agile environments. Very few of them mentioned the requirements in scrum, which has its own characteristics compared with other agile methods. Also, the requirements elicitation phase are discussed most while other important aspects of requirements engineering process, such as requirements description, requirements validation, etc. are rarely mentioned.

The case study in this Master Thesis is mainly focused on the RE process after the elicitation phase in scrum, i.e., requirements analysis, description, validation and management, which is almost absolutely executed within a scrum team and with product owners (POs) and developers involved. There are also some efforts made for the interaction between POs and developers.

3.2.4 Software Testing in an Agile Environment

This part is a bit out of the scope of my Master Thesis because its topic is about Requirements Engineering under agile methods. However, testing is tightly related with requirements of a project since testing in advance is another form of validating the requirements [6]. Therefore, I think it is necessary to introduce the testing work in an agile environment, and I assume that the readers of this thesis have had the knowledge about testing in traditional software engineering.

According to [6], in an agile development, the testers, developers and product stakeholders contribute to decide the user stories together. After it is decided which user stories to be implemented in the coming iteration, the testers will generate test scenarios based on these user stories and the scenarios will be split into test cases after they are approved by the business analysts. The test cases should have enough coverage and the developers' work is tested with the cases. There is no distinct testing stage - the implementation and the testing last during the whole iteration. Also, when there are more features implemented, the regression testing should be executed once every several iterations to ensure that the new functionality does not influence the existing features. When a release is approaching, an acceptance testing should eventually be done before the work is transferred to production.

There are also some challenges in the testing under agile methods [22]:

- The time of each iteration is fixed, which means that the testing cannot be extended even there are more defects found than the expectation.
- Testing in traditional software project process is based on documentation which is fixed in an early stage, but agile methods embrace changed requirements even in late phases and the documents will be updated consequentially. In this way, testing cannot be based on completed specifications.
- Agile development heavily relies on the intensive communication between the developers and the customers. Thus documentation even might not exist and the details of the requirements will probably be known by heart by the developers and the customers.
- Working software is the main evaluation of progress. So testing has to provide quality information continuously to ensure that the developers' code is working, and testing cannot be an independent phase in the iteration.
- Agile methods omit unnecessary work as much as possible to improve the efficiency. It is possible that testing is also removed since it does not add value like functionality immediately.

Testing in an agile environment is different from that in traditional waterfall process, but it is still based on requirements and can influence them if the testing work is not performed properly.

3.3 Scrum

Scrum is one of the agile methodologies. Similar to agile development methods, the process of scrum is iterative and incremental and the working style should be collaborative and communicative [32]. In this section, I will briefly introduce the development process, the rules and the characteristics of scrum method.

3.3.1 Process

According to Ken Schwaber, one of the founders of scrum [25], the process of software project development under scrum should be:

- Before the development, when a scrum team starts a new project or a new big release, there will be a **planning** which defines a comprehensive backlog list, where all the functionality requirements are logged. Based on the known backlog, the backlog items to be implemented in the new release are chosen, and the cost and the risk is estimated and analysed.
- The **architecture/high level design** takes place after the planning. The implementation of the backlog items is designed and the architecture of the system is refined to be adapted to the new environment and/or requirements.
- Then comes the development phase, where **sprints** happen iteratively. Usually a sprint lasts for one to four weeks. The backlog items in the current sprint are prioritized and are not allowed to be changed [32]. During a sprint, there are usually:

- **Daily scrum** which happens precisely on time and is time-boxed to 15 minutes. Every developer reports what he/she has done the last day, what he/she plans to do today and if there are any problems in the work.
- **Poker planning meeting** which is before a sprint starts. The team estimates the difficulty of working items with planning poker cards, with different numbers (from 0 to infinity) on each of the card. Nobody is allowed to show their card until they say "flip", so that everyone's estimation is not influenced by others'.
- **Sprint planning meeting** which is at the beginning of a sprint. The scrum team chooses what tasks should be done in the sprint and estimates the working time for these tasks [7].
- **Sprint review meeting**. The team presents to the stakeholders what they have completed.
- **Sprint retrospective**, where all the team members reflect on the past sprint, to see what has been done well and what can be improved in the coming sprint [36].

- The last phase is **closure** phase which happens when the management thinks that it is time for a new release. The integration, system testing, documentation etc are the tasks in this phase.

3.3.2 Roles

Here are the main roles and the main responsibilities of these roles in a scrum team [32]:

- **Product owner**: the one who speaks for the customers and makes sure that the team is working with what the customers need. A product owner should write customer-centric items, prioritize them and put them into product backlogs. During the working time, the product owner should sit with the developers and answer all their questions about the requirements.
- **Scrum master**: a scrum master is not a team leader, but he/she should remove the obstacles that prevent the team from releasing on time and make sure that scrum process is executed as it should be.
- **Team**: the group of people who are responsible for the development.
- **Customers**: participates in the tasks related to product backlog items for the system.
- **Managements:** makes the final decision and participates in the goals and requirements settings.

3.3.3 Characteristics

The scrum method has the characteristics of [25]:

- Both the planning phase and the closure phase are linear, and what should be done in these phases are clearly defined
- The process in a sprint is undefined and unpredictable. The risk management should be done on each iteration when the flexibility is maximized.

- Different with the planning or the closure phase, sprint is nonlinear, and the process knowledge is not always explicit. Sprints are adopted for the evolvement of the product.
- The deliverable of the project can be changed any time during the planning and sprint phases until the closure phase.
- The deliverable depends on the environment during the project phase.

4 Case study

4.1 Background of the organization

This Master Thesis is conducted in the company Sony Ericsson Mobile Communications, Lund, Sweden.

4.1.1 Background of SEMC

Here is a description from the official website of the company:

"*Sony Ericsson Mobile Communications was established in 2001 by telecommunications leader Ericsson and consumer electronics powerhouse Sony Corporation. The company is owned equally by Ericsson and Sony… undertakes product research, design and development, manufacturing, marketing, sales, distribution and customer services. Global management is based in London, and R&D is in Sweden, UK, France, Netherlands, India, Japan, China and the US.*" [37]

"Sony Ericsson Mobile Communications is a global provider of mobile multimedia devices, including feature-rich phones, accessories and PC cards. The products combine… technology with… applications for mobile imaging, music, communications and entertainment." [37]

4.1.2 Background of Test Tools & Support Section

There was reorganization in mid-May 2009, two weeks before when I finished my Master Thesis work at SEMC. Figure 4.1 shows the organization when the thesis started: The Test Tools & Support Section, where my Master Thesis was carried out, was one of the sections under Global Functions Department, which was under Creation & Development of Sony Ericsson. The section manager, Anders Nyberg, was my supervisor.

There are three teams in the section: BRAT (Batch Remote Automate Testing), QC (Quality Center) and a support team. BRAT team works with developing and maintaining BRAT, the test automation tool used for stability testing within SEMC, and QC team works on the implementation of HP's Quality Center. My thesis work was only focused on these two teams.

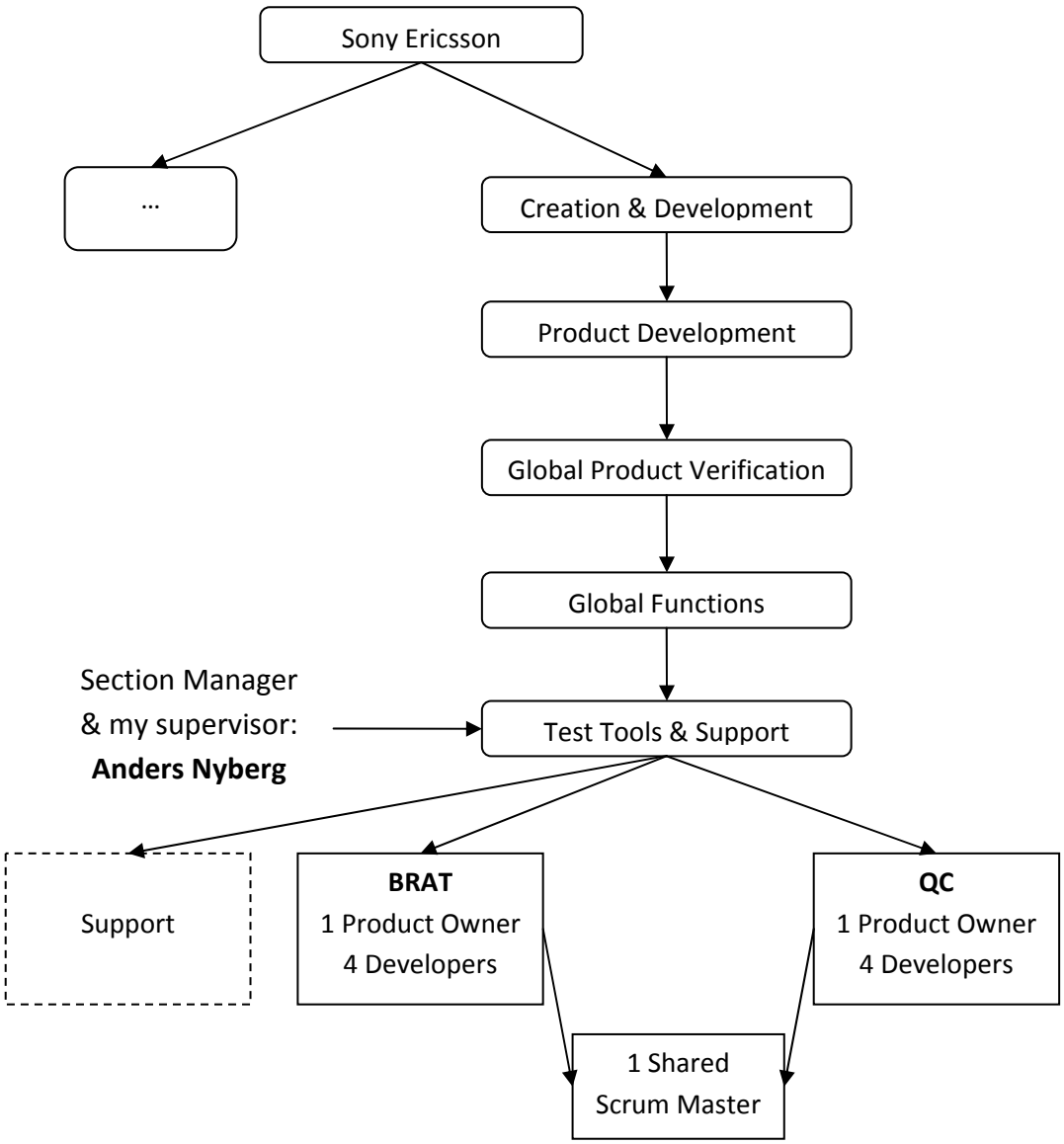

Figure 4.1 Organisationg of SEMC

4.2 Teams and Products

Both of the teams work on the products which are the tools for internal use within Sony Ericsson. Thus the customers are also employees in the same organization.

4.2.1 BRAT

BRAT team consists of six people, including four developers, one product owner and one scrum master, who is shared by both teams. The developers mainly work on the BRAT tool and sometimes they need to do some support work of other test tools in the section. The team has worked with scrum for about one and a half year and is very familiar with this method.

The test tool BRAT, the team's product, is an application that uses a 3rd party script language to execute test scripts. A user can create test cases in BRAT, and the tool will be able to simulate a user and run the corresponding test cases. Then the user can check the logs to see the results of the testing execution.

BRAT team works with the implementation and the maintenances of the tool. The team had released version R3A041 and was continuously updating the product and working on the maintenance when I started this thesis.

4.2.2 QC

There used to be five developers in the QC team, but one of them left in April. Same as BRAT team, they have one product owner and share the scrum master with BRAT team. The team works on the QC tool and also needs to do some support work occasionally. Some developers used to work with traditional waterfall process before the team was reorganized into the section in October 2008. Two of the developers have the experiences of working in an agile environment. Compared with the product owner of BRAT, the PO of QC is newer to the position and thus is not very familiar with some of the add-ons of QC, which the team was working on.

The team is working on the extension of HP Quality Center (QC). HP QC is a web-based test management tool which helps to maintain a project database of tests that cover all aspects of an application's functionality. A user can log the information such as use cases, detailed requirements, test cases and defects in QC. The status of the test cases (pass/fail) and the defects can also be traced through the tool. Also it supports the functionalities of importing/exporting information from Microsoft Word and Excel.

The work that QC team handles is mainly about the extension of some functionality based on HP Quality Center. Same as BRAT team, when I started my MSc Thesis they had released version QC 9.2 of the product and was working on the extension of the new functionality.

4.3 Current situations

The teams mainly follow the approach of scrum, i.e., they have the roles of scrum master, product owner and team players. However, there are no tester roles in either of the teams. Developers are responsible for the testing. They execute heuristic testing on each other's implementation work and run the system test at the end of a sprint. More details about the testing in the teams will be discussed in section 4.3.6.

4.3.1 Product process

As shown in Figure 4.2, both teams follow the process of:

i) Before a sprint, the PO collects the primitive requirements from the meetings with customers, the e-mails from customers or the defects description updated in DMS (Defect Management System, a system that is used for logging system defects, will be explained later in section 4.3.7).

ii) The PO interprets the customers' requirements into work items and logs them into backlogs.

iii) Poker planning meetings are held before a sprint starts. The developers discuss the details of the requirements, and estimate the complexity of a working item with planning poker cards [30].

iv) The PO decides the items to be finished in the following sprint according to customers' requests and the product's situation, and gathers the prioritized items into one sprint's block in backlog.

v) At the beginning of a sprint, the developers discuss the detailed tasks of each working item with the PO and estimate the working hours of the items.

vi) Based on the requirements, developers write and/or update the use cases before the implementation and log them into QC.

vii) When one single requirement is implemented by a developer, another developer executes heuristic testing based on the test cases.

viii) The developers execute the system testing together when all the tasks are implemented in the sprint.

ix) The release of the sprint is presented to customers in sprint review.

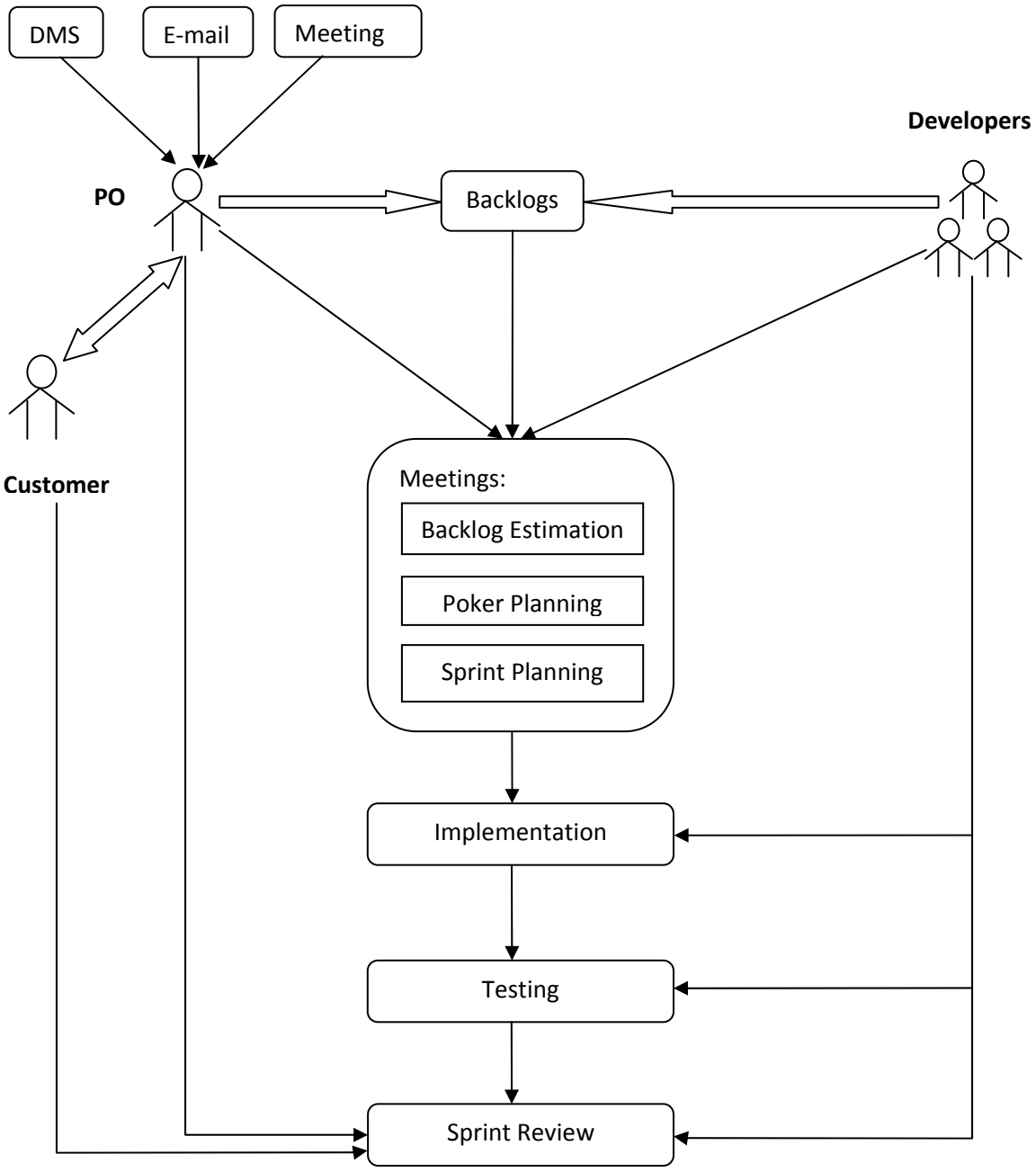

Figure 4.2 Working process of BRAT Team and QC Team

The Master Thesis is required to focus on the interaction between the PO and the developers, i.e. the process described above except for the first step.

4.3.2 Meeting composition

Intensive communication within the team is one of the typical characteristics of scrum. Both teams have the following meetings in the whole process of a sprint. (The overall meeting

compositions have been introduced in section 3.3.1, but here is the practical case of the BRAT team and QC team, and there are some slight variations):

- **Poker planning**
  - **Participants:** developers, PO, scrum master.
  - **Content:** developers and the PO discuss the new and updated items in the backlog, including the details, the reasonability etc. The complexities of implementing each item are also estimated by the developers with planning poker cards. The scrum master records the discussion content and the complexity points into the backlogs.
  - **Aim:** the PO gets to have clearer ideas about the requirements and thus can prioritize the working items in backlogs.
  - **Time:** in the middle of the sprint which is before the one that the poker planning is relevant to.
  - **Frequency:** once a sprint.
  - **Time length:** 2 hours.
- **Sprint planning**
  - **Participants:** developers, PO, scrum master.
  - **Content:** developers discuss the implementation tasks under all working items which are planned to be finished in the following sprint. Also they estimate the working hours of each item. The scrum master records the added tasks, the numbers of working hours and any content related to the requirements.
  - **Aim:** to plan the coming sprint in details.
  - **Time:** in the beginning of the sprint.
  - **Frequency:** once a sprint.
  - **Time length:** 4 hours.
- **Daily stand-up meeting**
  - **Participants:** developers, scrum master and sometimes PO.
  - **Content:** the scrum master prints out the Product Burndown Chart (see Figure 4.3) that the team has achieved until the day before. Every developer reports the work he/she has done the past day, any problems he/she has encountered and the plan for the current day.

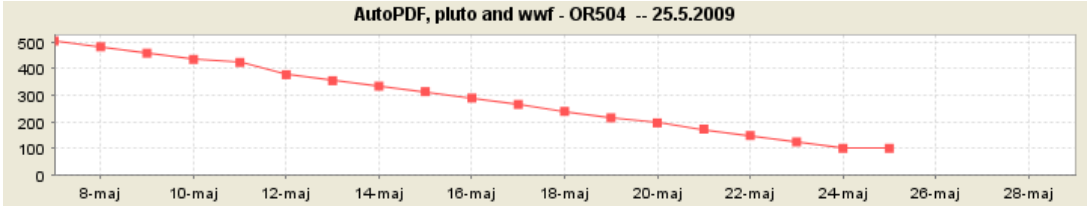

Figure 4.3 Burndown Chart, the x-axial refers to the dates in a sprint and the y-axial means the remaining working efforts of the sprint

    - **Aim:** to let the scrum master be aware of the process of the project, and to let everybody in the team know what others are doing and the problems the team has now.
    - **Time:** every morning during the sprint.
    - **Frequency:** once a workday during a sprint.
    - **Time length:** 15 minutes. If there is any big issue to be discussed, the team should talk about it after the daily meeting.
- **Sprint review**
    - **Participants:** developers, PO, customers.
    - **Content:** one of the developers presents what has been implemented during the last sprint. The customers ask questions about the demonstration and the developer who is responsible for the relevant functions answers the questions. The customers might also have some more requirements based on the current result of the product.
    - **Aim:** to show customers the new updates of the project and to get their feedback on it.
    - **Time:** at the end of the sprint.
    - **Frequency:** once a sprint.
    - **Time length:** 1 hour.
- **Retrospective**
    - **Participants:** developers, scrum master.
    - **Content:** developers recall what has happened during the last sprint. The changes, anything out of expectation and anything that they think worth being mentioned are noted down on a whiteboard. The team then discusses the listed events and evaluates them as good or bad. The scrum master records the content of the discussion.
    - **Aim:** to conclude last sprint and learn some lessons from it.
    - **Time:** in the end of the sprint.
    - **Frequency:** once a sprint.
    - **Time length:** 1 hour.

Here is sequence of the meetings in the linear time is shown in Figure 4.4.

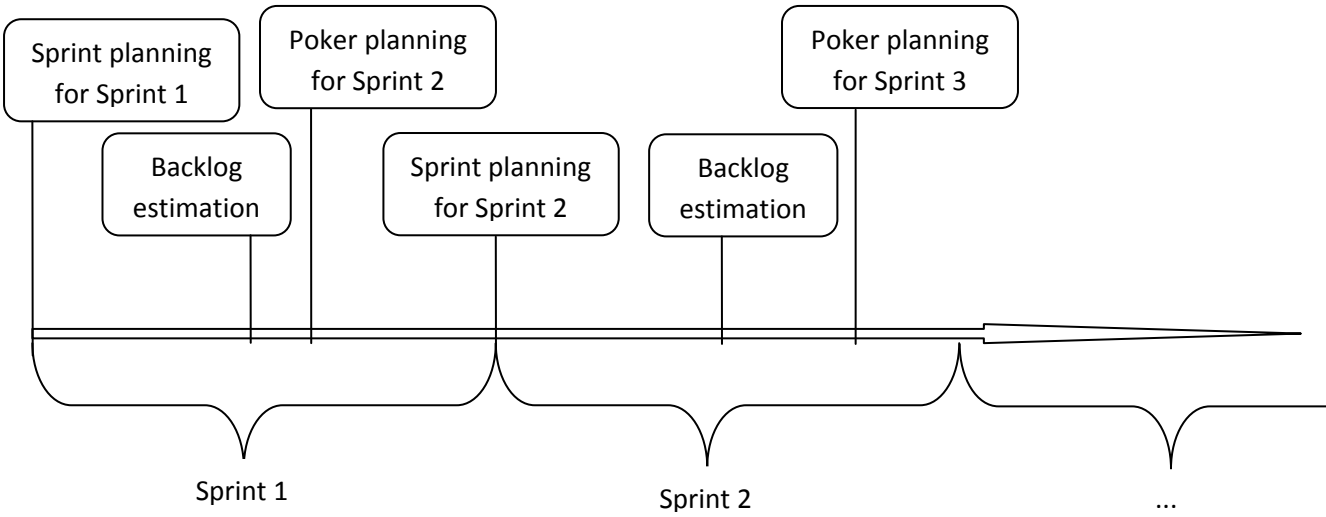

Figure 4.4 Meeting composition and sprints

4.3.3 Sprints distribution

Different with the general definition of scrum, the teams do not have continuous sprints to implement the incremental requirements of the products. Instead, they usually have alternant sprints of new requirements implementation and old defect fixing. I.e., they work on new requirements for one sprint and fix bugs for the next sprint. Also, sometimes there is a mixed sprint which has both new implementation and bug fixing work.

The new-requirement-implementation sprint usually lasts for three to four weeks depends on the numbers and the complexities of the requirements, while the length of a bug fixing sprint is more random. When it is time for such sprint, the teams will combine scrum with the Kanban method [1], which means that all the defects to be fixed are listed on a whiteboard, and the teams continue fixing them one by one until all the tasks are finished. Meanwhile, the daily stand up meeting, which is from scrum methodology, is also adopted. During the stand-up meetings, the developers remove the tasks they have finished from the whiteboard and decide what tasks they will work on that day. The length of this kind of sprints is usually unpredictable since the sprint will not end until all the defects are fixed.

Besides these two kinds of sprint, sometimes there can be a short sprint which is intended to solve some urgent issues depends on the current situation. The sprint usually lasts for one to two weeks in this case.

4.3.4 Customer Involvement

As I have mentioned in section 4.2, both of the product are internal tools so the customers are also the employees at SEMC. The two teams have different customers and the situations of their customers are not the same:

- **BRAT team:** there are some local customers who sit in the same building as BRAT team, so it is easy to get access to them. The team usually demonstrates the new implements to these customers and gets their feedback in time during sprint reviews. The other

customers are working in different places in the world, such as US, China, Japan, India etc. The PO gets the requirements and the feedback from them via e-mails, telephone conferences, DMS (will be explained in section 4.3.7) etc. The end users of BRAT are testers within SEMC, and there are about 150 of them.

- **QC team:** similar to BRAT team, QC team has global customers including the ones who are located in Lund. The main ways for the PO to communicate with the customers are having meetings and sending e-mails. Other methods such as telephone call are also used. However, the team also develops some functions which aim to all the general users who use QC, and there are about 1500 such users. In this way, it is hard to identify any specific customers and the PO should speak for them.

As described in section 4.3.1, the POs get primitive requirements from the customers first. Then they interpret the primitive requirements collected from meetings, emails or conference calls roughly into working items in the backlogs. The defects logged in DMS by customers are also copy-pasted into backlogs as the requirements for the teams.

In the meetings of poker planning and sprint planning, developers look into the roughly-interpreted working items and more details are decided and added under the items. However, sometimes, especially in the QC team, the PO is not very familiar with all the working items of the team and there might not be representative customers, which makes the PO more or less passive during the discussion. Sometimes none of the team members, including the PO himself, is sure what the customers really want. The team thus can only “design” the requirements by themselves.

4.3.5 Documentation

The largest requirements documents that both teams have in hand are the backlogs. All the historical requirements, the relevant working items, the tasks and the details are logged there.

The developers generate use cases based on the requirements of the products. The use cases are usually the main success scenarios, and the detailed steps of how a user should go through these scenarios are listed one by one under each case. The cases are logged in HP Quality Center, and the developers need to check on them when they execute the heuristic testing spinning around the use cases.

One thing should be specifically mentioned here is that there is not any formal documentation about the requirements in either of the team. This will be a potential problem for the long-term maintenance of the products.

4.3.6 Testing

Neither of the teams has full-time tester, so all the testing related works are done by the developers. They first test others’ implementation work once there is some functionality finished. When all the requirements in the current sprint are implemented, every developer participants the system testing to cover as many details as possible.

The teams used to do script-based testing, i.e., they generate test cases first and then run the cases according to the steps described in these cases. Now they have switched to heuristic testing, i.e., the developers do not need to write test cases. Instead, they test the functionalities according to the use cases. One use case is mapped to one big test case. When the big test case is passed, the details around the main success scenario, such as some negative cases, the boundary values will also be tested. This save the developers from the long documentation of detailed test cases, but there is no proof to assure that the testing will cover everything before the release of this sprint. Furthermore, only the functions of the products are tested. The non-functional aspects, such as performance or security, of the products are not paid any attention to. There will be more details about the problem analysis in next chapter.

4.3.7 Tools

Currently there are three tools relevant to requirements used by the teams:

- **Scrumworks:** it is the most often used tool where the backlogs are stored. The POs use it to log requirements and relevant working items and the developers update their process through it. Scrumworks is also used during backlog estimation, poker planning and sprint planning. The teams discussed the tasks displayed in the Scrumworks and the scrum master adds details into it.
- **Quality Center (QC):** as introduced in section 4.2.2, developers log use cases and relevant steps into QC. They need to check the cases while executing the testing and the pass/fail result of the test cases are also logged in QC. One interesting thing is that QC team was working on the add-ons of HP QC. At the same time they were using the existing HP QC as a tool as well.
- **Defect Management System (DMS):** it is an internal tool at SEMC. In most cases, customers log the defects in DMS which tracks where an issue is in the process. The POs transfer the newly reported defects from DMS into Scrumworks as requirements for the teams.

5 Analysis

5.1 Reflected problems

Some problems of the requirements engineering process in both team reflected during the case study time. In the following part of this section, the categorized issues, which I found during the observations and the interviews, are listed out. The consequences of these problems will also be analysed.

5.1.1 Requirements not detailed enough

I attended the poker planning and sprint planning meetings of both teams and observed the way that the teams discussed the working items. One of the revealed problems was that some of the requirements did not have enough details. QC team was more serious in this aspect.

For example, some description of an item was too simple, or there was even only a title. From the interviews, some of the developers said that they knew what should be done by heart, while others admitted that it would be better to have more details. Another example is that sometimes one of the developers might be not quite sure about what the working item meant, so others would explain it to him. Once a developer asked what kind of testing steps should be included in a task called “system test” and others answered him. However, the answer was not noted down, and the task still had only one title.

Without enough details, there can be misunderstanding of to which extent the requirements can be regarded as “pass”, and it would be too late if the misunderstanding really happens when the team starts the implementation phase. Since there is someone who perhaps will forget the content of a working task/item, there can be others who are not sure about other requirements either. If the details are not added into the description of the task, the team will probably forget what the tasks are about after a long time, and it will be hard for them when they need to check for some information. Furthermore, it will be difficult for other people to read the requirements description if later on the product development is taken over by others.

In this way, it would be better if the discussion content and all the details relevant to the requirements are written down. Also, sometimes maybe most of the developers are sure about what the working item, which has only one sentence, means, it is still better to add more description of it.

5.1.2 Mixed formats of requirements

There are different kinds of working items in backlogs, such as user-related requirements or the tasks which are purely about technical implementation. However, the descriptions of most requirements are in “random” formats. Some of them showed the rough steps of completing a scenario, some were a paragraph of texts and some were just a segment of a customer’s e-mail. The descriptions were far from polished, and usually the developers needed a relatively longer time to read the description under the working items and then figured out what the item was talking about.

Without a consistent format of requirements description, the team could hardly get the information which they needed in a short time. Also, what the customers really want might not be included in the requirements even though there are long texts in the description. In this case the team risk of not understanding what users want and implementing some functions which are not the real needs of customers.

5.1.3 Unclear bug description

As I have introduced in section 4.3.3, there are two kinds of sprints in both teams. One kind of sprints is bug fixing, so the requirements of that kind of sprints are mainly about the defects reported by customers.

Similar to section 5.1.2, the description of defects in backlogs is unclear, and the situation in QC team was more serious. Usually the product owners copied and pasted the relevant defects from DMS into the backlogs without going through the descriptions in advance.

From the interviews I heard some complaints that the customers often listed out what they wanted the developers to do without telling them the reasons why they wanted these things to be done. The developers could not understand the ultimate intents of the customers and often misunderstood the tasks they were asked to implement. As a result both the time and the resource were wasted.

5.1.4 QC team not always understanding the requirements

From the observations and the interviews, I found one of the biggest problems of QC team was that sometimes the team, including the product owner was not sure what the requirements meant.

Compared with the PO in BRAT team, the PO of QC team had less experience of this position. Also, he was new to the project so he was not familiar with all the functions of QC as mentioned in last chapter. Some customers of QC were also harder to be accessed. All these reasons led to the result that the PO of QC was passive during the discussion on the working items.

Very often, it was the developers who talked about the requirements actively while the PO just sat there and listened. Sometimes when the team encountered a requirement which they didn't quite understand, the PO might have no idea either. In this way, the situation became like:

- the team tried to guess how the requirement should be like
- they wrote down their ideas
- the team implemented the work according to the uncertain thought

However, a product owner should have complete knowledge and the power to make decisions. If he cannot implement the work well, the effectiveness of agile methods might be reduced because he is tightly coupled with customers [11]. Once again, the problem made the team to risk programming something which might not be the customers' needs since it came from the developers' guess.

5.1.5 No formal documentations

After talking with everybody in the teams, I realized that there was not any formal documentation of the requirements in either of the teams. The teams' most complete records of requirements are the backlogs, but they are far from formal or well-documented.

The mentality of agile methods is iteratively changed requirements and no need to write all the fixed requirements in long and tedious documents at the beginning of the project. However, it does not mean that there should be no documentation at all in a scrum team [19]. The lack of documentation might cause long-term problems. No one can be sure that the developers will remember all the details of the products for ever or they will always work in that project team. Without complete and formal requirements documentation, it would be hard for the teams to do the future maintenance work if some members leave the team. Also, the work would be slowed down if a new member knows about the project by asking questions to other members rather than reading a complete document when he/she joins in the team [17].

5.1.6 No non-functional requirements (NFRs)

From the backlogs I could not find any non-functional requirements. While asking the developers and the product owners, they also admitted that they had nearly never handled NFRs unless there was a relevant defect reported from the customers, for example the performance of some function was not good enough.

Maybe the teams do not think that they need NFRs since they currently focus on implementing new requirements and testing them. But for a mature product, non-functional requirements must not be missed. Actually, it should be handled from the very beginning of the project [10]. If there are only functional requirements when the sprint is planned, nobody will pay attention to NFRs and the testing will be only spinning around the functionalities. As we know, the earlier a defect is found, the lower the cost of fixing it will be. It is the same for NFRs. It will be too late when the NFR related defects are found by customers after the release of the product. Furthermore, the cost of fixing defects such as performance issues is usually higher than fixing an error of functionalities. In this way, the NFRs should be dealt with in a project as soon as possible.

5.1.7 No full-time testers

The team composition in section 4.2 shows that there was no full-time tester in either of the teams. The developers execute the testing on others' implemented functionalities and do the heuristic testing based on the use cases.

Nevertheless, during the interviews some developers told me that when they finished their programming work, they had been quite tired and would probably not make much effort on testing, since they were "developers" rather than "testers".

They believed that it was not professional enough to ask a developer to do the testing work and the tasks should be assigned to the right roles. Also, heuristic testing means that there is no test case designed before the implementation starts. However, writing test cases according

to the requirements is another means of validation [26], because the test case designer needs to think through the requirements and make sure that all of them are testable. In this way the teams lose a good chance of requirements validation. What's more, without full-time testers, the developers' work might not be tested in time and the bugs might be found later, which could increase the development cost. Figure 5.1 shows the relationship between the cost of fixing bugs and the phase in a software project process where the bugs are found [13].

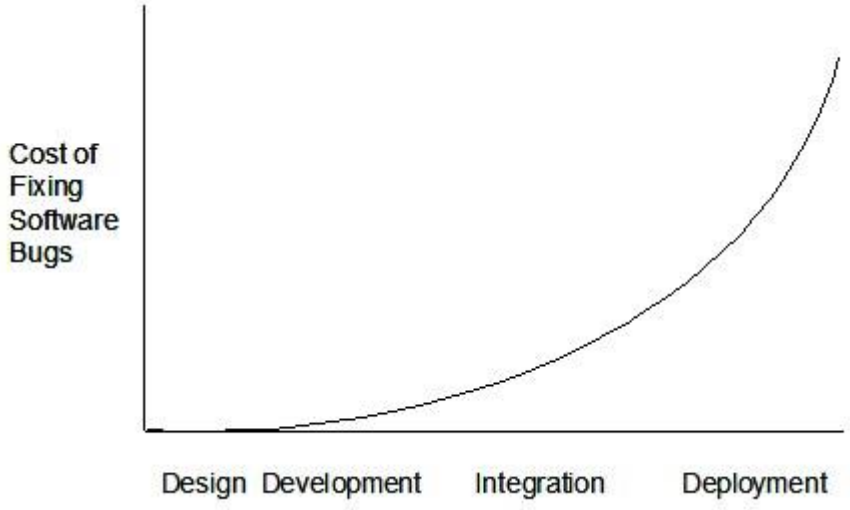

Figure 5.1: The growing cost of fixing bugs. The longer you wait, the more you'll pay.

5.1.8 Developers rather than PO demonstrating the product to customers

At the end of most sprints, especially the sprints which focus on the new requirements, there will be sprint reviews that both the teams and the customers will attend. During the review one of the developers introduces the release of that sprint and showed the new functions to the customers.

But I think it is inappropriate to let a developer to do the presentation, since a developer is responsible for only a part of the requirements while the PO has a whole picture of the product. Also, the PO might not be very motivated in knowing more about the product if one of the developers handles the presentation and the others are responsible for answering the technique related questions during the meeting.

5.2 Possible improvements

Based on the problems analysed above and the different situations of BRAT team and QC team, I figured out some possible improvements for each team respectively.

5.2.1 BRAT team

- **User stories**

  When a PO describes a requirement with user stories, he/she will have to think from a user's perspective of view. User stories usually represent a very small part of a

scenario. In this way the requirements become smaller and more potential details might pop up automatically when the PO is thinking about the scenarios. It is also easier for developers to know what customers want while reading a user story. Since user stories are of smaller granularities compared with use cases, when heuristic testing is executed, they might be helpful for the developers to check more details of a scenario. Furthermore, the format of user stories should be acceptable for customers since they are described on behalf of an end user.

Recommended implementation:

- After figuring out the main success scenarios with the customers, the PO (sometimes with the customers) writes down the use cases in Quality Center and logs user stories into Scrumworks in the format of:

  As a [role] the user wants to [function] so that [rationale]

- The team discusses the relevant requirements together and adds the programming tasks and makes the estimation hours on the basis of the user stories.

- **Test designer role**

  With some, or even at least one, full-time tester(s) in the team, when the requirements are discussed during the meetings, the tester will start thinking how to generate the test cases based on the requirements. Some description which is not clear enough to be tested might be found out, and the testability of the requirements is validated in this way. When the developers start the programming right after the sprint planning, the tester will begin to design test cases, which is a more thorough round of requirements validation. Also, once there is some functionality implemented, it will be tested at once following the test cases and the defects will probably be detected at first hand. When there are full-time testers in a team, the quality of the product might also be improved since the developers are able to focus on their implementation work and the testing will be done thoroughly since it is the testers' responsibility.

  Recommended implementation:

  - Spare one of the developers in the team as a test designer.
  - The tester discusses the requirements with the team during the meetings, but from the testing's perspective.
  - The tester starts generating the test cases right after the sprint planning.
  - Once there is some functionality implemented, the tester tests it according to the test cases and reports the bugs in no time.
  - When all the implantations are finished the whole team executes the system testing together.

- **Requirements template**

When a template is adopted to describe the requirements, the formats of the requirements description will be more consistent, and it is easier for developers to get the information they want in a short time. If the template includes all the necessary aspects of a requirement, such as user stories, implementation tasks, done criteria, etc., the PO will have to think through all the necessary parts and the developers will know what the customers want and everybody in the team will have a common agreement of what "pass" really means. Also, some missing elements might be discovered the in time and the PO gets to contact the customers before the meetings.

Recommended implementation:

- When the PO interprets the primitive customers requirements into backlogs, the user-related requirements should be formalized into the following template which includes the user story suggested above:

Title

- **User story:**

  As a [role] the user wants to [function] so that [rationale]

- **Description**

- **Implementation tasks**

  - Task 1
  - Task2
  - …

- **Done criteria**

  - Criteria 1
  - Criteria 2
  - …

Figure 5.2 Template of requirements description

- During the poker planning, the developers discuss more details about the requirement and the implementation tasks are listed out under the user story.

- **Defect description**

As some developers said, sometimes they did not know what the customers exactly wanted in their defect reports. Similar to the requirements template, the defect description should also have some format which ensures all the necessary aspects such

as the customer's expected result of some behavior that developers need are included. The content of the description will also be formalized and the developers will be able to get the necessary information more quickly in this way.

Recommended implementation:

- The PO goes through the defects in DMS before copying them into Scrumworks.
- All the defects description in Scrumworks should be in the form of:

Title
o **Reproduce steps** • Step 1 • Step 2 • ... o **Expected result** o **Actual result** o **Additional information**

Figure 5.3 Template of defect description

- The PO contacts the relevant customer if he finds something unclear and then makes up for the missing parts. Also, screenshots can be attached into the defect description if necessary.
- The developers discuss the defects during poker planning and sprint planning.

- **Splitting big working tasks into smaller ones**

  Sometimes there are some tasks estimated as 15 or even 20 hours, while the description is not detailed enough. If a large task is separated into smaller ones, then more details will probably be considered and the estimation of the working hours might be more accurate.

  Recommended implementation:

  - In sprint planning, the tasks that need more than 6 hours, which is defined as the workload of one man-day in the team, should be split into smaller ones. No task should be longer than 6 hours.
  - The implementation steps should also listed roughly under each task.

- **Non-functional requirements**

Non-functional requirements are also a part of a product. It must not be missed if the project is in a long-term plan. The earlier the NFRs are handled, the smaller the cost of future maintenance will be.

Recommended implementation:

- The PO and the developers discuss and make out a checklist of the categories of NFRs according to Figure 3.2 in section 3.1.1. There is no need to enumerate all the types, though. The team should consider their practical situation and illustrate the aspects they think necessary.
- Following the checklist, the PO tries to get different sorts of non-functional requirements from customers first. If the customers are not sure what the specific requirements should be, even a rough idea will be fine.
- According to section 3.1.1, non-functional requirements can be described in the form of user stories so that the reasons of having such a NFR will not be forgot later when people look back onto it [38]. However, the team does not have to stick with this form. They can express the NFRs as they feel comfortable. But the NFRs description should be quantitative so all the requirements can be verified. The metrics in Figure 3.3 can be a reference. If the team decides to use user stories, the template in Figure 5.3 can also be adaptive in this case.
- The NFRs should be dealt with at the same time when user stories are scoped with customers [10].
- The developers also discuss the potential constraints they can think about during the poker planning and/or sprint planning meetings.
- The PO logs the NFRs into backlogs, separated from functional requirements, and the team makes sure that all of the non-functional requirements can be verified/are testable.

- **PO doing the presentation**

A product owner should be a bridge between the customers and the developers. When it is time to demonstrate the team's work, it is the PO's responsibility to do it in the sprint review. With the presentation task in mind, the PO might be more motivated to know more about the requirements and think from a user's perspective of view, since the audiences are customers. It is also helpful for the PO to draw the whole picture of the product better.

Recommended implementation:

- The presentation to customers in sprint review is done by the PO.
- When customers have technique related questions, the developers should answer them as the backup of the PO.

5.2.2 QC team

Besides the improvements listed above, there are two more possible solutions for the QC team. The "PO doing the presentation" improvement is also altered based on the team's practical situation.

- **PO sending the requirements guesses to customers**

  Since the PO of QC team is not quite familiar with the product or the requirements, it is hard for him to learn all the necessary knowledge in a short time. The situation of the customers cannot be changed either. But interaction with customers is one of the key factors for project success [10], so if it is difficult to get enough information from the customers before the requirements are discussed with in the development team, it is better to have some feedbacks afterwards than doing nothing. The team should try to find out more questions and let the PO send the questions to customers and get their feedback if possible, so that the misunderstanding of the requirements might be reduced.

  Recommended implementation:
  - When there is something unclear for both the PO and the developers, the guess of the vague part should be written down during the meeting.
  - Right after the meeting, the PO sends the guesses to relevant customers and gets their feedback.

- **Glossary**

  From the interviews some developers mentioned that there was misunderstanding within the team due to the different terms used for the same object. A glossary list can be helpful in this case. When the terms of all the objects are confirmed and listed out in a document, this kind of misunderstanding can be avoided. It is also easier to introduce the product to people outside the team with the consistent terms.

  Recommended implementation:
  - The developers and the PO sit down and list out all the possible objects and confirm their names, to make the glossary list recorded in a document.
  - The PO then sends the glossary list to the customers.

- **PO doing the presentation**

  The reason of letting the PO of QC team do the presentation during the customer reviews are the same as those mentioned in last section. Furthermore, it is a good way to make the PO to get familiar with the requirements sooner. However, it might be difficult to ask the PO to do the demonstration at once. It is more reasonable to execute this item 2-3 sprints after the pilot.

  Recommended implementation:
  - Give the PO 2-3 sprints' time to "learn" more about the requirements, and then let the PO demonstrate the new results to the customers in a sprint review.
  - When there are technique related questions from the customers, the developers should answer them as the backup of the PO.

5.3 Selection Consensus

All the possible improvements introduced above were generated based on the literature studying, the observations and the interviews. It might still be away from the teams' practical condition. The developers and the product owners are more experienced and know their products best, so they will probably have better suggestions on these improvements based on their working experiences. In order to get more feedback and to make sure that everyone would agree on the pilot content, i.e., the improvements, I held a meeting with both teams respectively to get their agreements on some or all of these improvements.

5.3.1 BRAT Team

The Selection Consensus was held within BRAT team first. Both the developers and the PO were very willing to try out the pilot and decided to keep the following improvements items without any change:

- **User stories**
- **Non-functional requirements**
- **PO doing the presentation**

The team would also like to try the "**done criteria**" part of requirements templates. They thought that the necessary implementation tasks had been listed out and they would ask if there was anything unclear to them, but the developers believed that the "done criteria" would help them to think about testing work.

The items they would like to postpone or not to try are listed below and the reasons are described under each item:

- **Test designer role (postponed)**
  The developers liked the idea but both developers and the PO all thought that the cost would be too high, since the development time would be longer if one of the developers would work as a tester and the workload not changed. However, the team agreed that there should be full-time testers later if the budget allowed.

- **Defect description (not try)**
  The team would not try it. The PO thought that the workload would be too high if all the defects should be read through before they were logged into backlogs. Also they believed that the developers were fine with the current defect description although it would be better to adopt some templates.

- **Splitting big working tasks into smaller ones (not try)**
  Both the developers and the PO thought that not all the tasks could be split. The current max hours of each task were 12. If there was any task estimated as 12+ hours, it meant that the team was not sure about the task and would need more time to research the relevant problems. Usually such kind of tasks would be postponed to later sprints.

5.3.2 QC Team

The Selection Consensus with QC team was a bit different from that with BRAT team. The PO happened to have another meeting at the same time, so I discussed the possible improvements with the developers first, and then discussed the output of the Selection Consensus with the PO later.

Same as BRAT team, the developers were willing to try out a pilot. The items that they would like to try were:

- **PO sending the requirements guesses to customers**
- **User stories**
- **Requirements template**
- **Defect description**
- **Glossary**
- **Non-functional requirements**
- **PO doing the presentation**

Among the items above, the developers thought that **non-functional requirements** should be discussed within the team first, since sometimes customers were not familiar with techniques and might propose some unrealistic requirements. When there are some rough ideas generated from the team, the PO should then send the ideas to the customers and get their feedback.

The items they would like to postpone or not to try are listed below and the reasons are the same as BRAT team:

- **Full-time tester (postponed)**
- **Splitting big working tasks into smaller ones (not try)**

5.4 Pilot

We decided to have one sprint pilot in each team after the Selection Consensus. As I have introduced in section 4.3.3, the content of the sprints depends on the current situation of the teams and is more or less random. When the planned time of the pilots came, QC team decided to do some cleanup work. There were not many requirements related to their work in that sprint, so we had to give up the pilot. I will put some of the pilot items that the team has agreed to try in their future suggestions.

The BRAT team also had a special sprint during the pilot time. Luckily most of the pilot items could be tried in that sprint. The team had a short sprint, which lasted for only one week, to fix some defects reported by the customers. During the short sprint, they tried

- **User stories (in poker planning)**
- **Done criteria (in poker planning)**
- **PO doing the presentation (in sprint review)**

The sprint was mainly about defect fixing so there was not much to do with non-functional requirements. But the team said that they would like to try NFRs in the future whenever any NFR could be derived.

6 Evaluation (result of the pilot)

In order to evaluate the pilot which was tried out in BRAT team, I collected the feedback from the team by discussing with the developers and the PO separately. They commented positively on most of the pilot items and proposed some useful suggestions:

- **User stories**
  Developers were not quite sure about user stories since not all the requirements in the pilot sprint were user-related. But they believed that user stories must be used in the requirements concerning with users, especially the new requirements that were user-related, so that they would know what the customers wanted to do for some specific function. They also thought that user stories were more helpful for test-case-based testing than the heuristic testing they were following.

  The PO had the same thoughts about the user stories. He thought that user stories were more adaptive to user-related and new requirements.

- **Done criteria**
  The developers welcomed the done criteria very much. They illustrated the good parts of using them:

  a) The team was forced to think through the requirement precisely, and more details were considered so that the estimation of working hours could be more accurate.
  b) The PO had to think what he really wanted and might find that the original requirements were not realistic enough.
  c) The testing goal became clearer because everybody knew what "pass" meant exactly.

  Besides these good parts, the PO thought that the "done criteria" should come from customers to the team so that the criteria would be more accurate. But there also could be a risk that the developers did not do anything out of the description of the done criteria since the PO "didn't say that".

- **PO doing the presentation**
  The developers like this idea because they could focus on their programming work without worrying about what should be said to the customers during the sprint review. They also believed that the PO would know the requirements better by doing the demonstration himself.

  The PO liked to do the presentation as well because he thought that he could have more communication with the customers in this way.

Besides these comments, the PO also thought that some of the developers talked much on the meetings while others did not talk unless they were asked. He thought that there should be a template of requirements discussion. The template should include the aspects of uncertainty, complexity, possible solution, best/worst case solutions, etc. The way of thinking

might be formalized in this way and the situation that some people talk a lot while others talk very little can be improved.

The PO said that the adaption of the items was limited and the team needed a longer sprint to try them. But both the developers and the PO in BRAT team said that they would be willing to using these pilot items, including the non-functional requirements in the future.

The pilot sprint ended in time without any tasks unfinished, and the team had some positive comments on the items. So I think that the pilot can be labeled as successful.

7 Conclusion

7.1 Future suggestions

Because of the limitation of the time and the random content of the sprints, only a part of the planned pilot items were tried out in a sprint by BRAT team. Based on the untried items and the feedback on the pilot, I propose some future suggestion for both teams in this section.

7.1.1 BRAT team

- Adopting **user stories** when there are user-related requirements. Since the aim of writing user stories is to let the developers know what customers really want, there is no point to write user stories when the requirement does not contain any user behaviours.
- Keep the "**Done criteria**" part of each requirement, but the PO should try to get the criteria first from the customers, so that they are more accurate and the PO and the customers share the responsibility of defining the pass criteria together.
- The **PO does the presentations** to customers in sprint reviews, and the developers should be the PO's technique backup.
- **Non-functional requirements** should be discussed in the future sprints. According to the suggestions from QC team, the developers and the PO should discuss the NFRs first and have some general ideas. Then the PO should get customers' feedback based on these ideas and polish the NFRs. When the project becomes more mature, NFRs will be needed sooner or later. In this way it is better to handle them as soon as possible so that the changes to the system will be limited to a lowest extent.
- If budget allows, the team should have at least one **test designer** who is responsible for the testing work. The process of designing test cases is also a way to validate the requirements. Furthermore, some defects might be discovered earlier with a test designer role in the team.
- There should be **formal documentation** of the requirements in the team. It was a problem I realized during the pilot. Agile helps the team to get rid of long and fixed requirements at the beginning of the project, but it does not mean that the team should have no document at all [19]. When a project team has worked on a product for a long time, a formal documentation which records the history requirements will help the team with future maintenance work or to trace the old requirements more easily if there have been any changes. It is also easier to explain the product to others in case there is no "old" member in the team. The team should start to work on the document right now and update it every two to three sprints.

Overall the BRAT team has been familiar with the scrum method and conducted it well. The interaction among the developers and between the PO and the developers is intensive and straight-out. The progress of the development is under good control in every sprint. They can work better with scrums if some or all of these suggestions are adopted in the future.

7.1.2 QC team

It was a pity that the pilot was not tried in QC team. However, I got many useful suggestions from the interviews and the Selection Consensus with the team. Based on the lessons learned from the pilot which was carried out in BRAT team, I hereby list some suggestions for QC team in the future:

The improvements should be tried with the same reasons as BRAT team:

- **User stories**
- **Done criteria**
- **PO doing the presentation (2-3 sprints later)**
- **Non-functional requirements**
- **Full-time tester**
- **Formal documentation**

Based on the condition of QC team, some other changes can also be helpful to improve the current situation:

- It is hard to influence how the customers discuss the requirements with the PO or to increase the PO's knowledge of the product's requirements in a short time, so the most possible way that the developers can do is to get more questions about the requirements and **have the PO to ask the customers**. It is slower than having complete requirements in advance, but better than implementing the functionality which is not the customers' real needs.
- It will be helpful for the developers if all the terms can be consistent. So if the team can sit down together and discuss the **glossary**, the misunderstanding caused by several names on the same object will probably be avoided.
- The developers complained they did not know what the customers really needed in the defect description and they wanted to have a neater description to read. Since the way that the team deals with the requirements is not very mature, **formalizing the defect description** is a good breakthrough to start with.

The QC team has not been very familiar with the scrum method, but they are getting used to it and becoming better and better. There is more space to improve the requirements engineering process in QC team than in BRAT team, so it is also easier to make some improvements in QC than in BRAT. Hopefully the situation will be mended with some of the suggestions tired.

7.2 Achieved aims

In section 1.3, I set some purposes to be achieved for the project. They are:

i) investigate the working process of the two teams in the test tools section at SEMC

ii) find out the problems the teams have in working with the current requirements
iii) design some possible improvements based on the discovered problems
iv) try out the improvements in one pilot for each team
v) evaluate the effects of the pilots in order to propose a suggestion on the future work of the teams

Here is the reflection of this thesis project according to the predefined aims:

i) Is fulfilled by the observations during the meetings and the interview to the scrum master and the product owners. The case study is described in details in Chapter 4.
ii) Has been performed by observations and the interviews as well. I interviewed the product owners of both teams and most of the developers and got to know what problems they had encountered. Also I found out the problems of the teams from their discussion on the meetings.
iii) Is one of the most important purposes of this Thesis Project. I figured out some possible improvements based on the discovered problems and the individual situation of the two teams. The teams discussed them and selected the improvements they thought acceptable. However, because of the factors such as limited budget, not all of the improvements were chosen to be tried. Also, the thesis project was asked only to focus on the interaction between the product owners and the developers, so there was very little access to the requirements elicitation phase, when the product owners should discuss the requirements with the customers. Maybe some more revolutionary solutions which could thoroughly solve the problems, that the teams have while working with the requirements, would be found if that phase was welcomed to be studied on.
iv) Can only be regarded as “half-achieved”. The pilot was not tried out in QC team as planned due to the special sprint which was temporarily decided. However, BRAT team tried most of the suggested improvements in a pilot sprint successfully.
v) Also “half-achieved” since only one of the teams tried out the pilot. The evaluation was mainly on the basis of the feedback from the BRAT team. Most of the feedbacks were positive and they proposed some useful suggestions for the future improvements.

7.3 Lessons learned

Most purposes of this thesis have been achieved and most of the feedbacks on the pilot, which was one of the most important goals of the paper, were positive. When reflected on the thesis project, however, I found that there were still some places which could have been done better, and here are the lessons learned from the project:

First, the biggest pity of the project is that QC team did not have the chance to try out the suggested improvements. In fact, a typical characteristic of scrum method is that requirements can be changed so the content of future sprints is unpredictable. Unfortunately, the planning of the thesis was not “agile” enough. Instead, it was rather “waterfall”. In this way, when the pilot time came and the team happened to have to do some other work which was not very related to requirements, the pilot could not be tried and the time would be up soon. So if there has been more time spared for the pilot, say the time for 2 or more sprints, then the pilot would probably be executed in QC team. Also, the BRAT team might have been able to try the non-functional requirements if there was more time left.

Second, the thesis was expected to improve the requirements engineering process of the teams in order to achieve the ultimate goal, i.e. to improve the qualities of the products that the teams were working on. During my case study, I found some problems relevant to not only the RE process, but also other aspects, such as testing. It has been pointed out that if the quality is intended to be achieved, a more overall approach should be adopted, rather than just concentrating on the RE process [29]. As mentioned in section 1.4, there were another two students doing their MSc Thesis on the testing process. If we have cooperated more, maybe we would have found some solutions, or at least some suggestions, to the testing issues. Also, the thesis was required to be just focused on the interaction between the POs and the developers. In my opinion, nevertheless, if the teams need some more constructive proposals, it would be better to investigate the RE process from the requirements elicitation phase (i.e., when the production owners collect the requirements from the customers). The more comprehensively the RE process is understood, the more helpful the suggestions could be.

Finally, the tools used for the requirements such as Scrumworks were not evaluated. Usually a good tool can probably help people work more efficiently. If more efforts have been made on looking for an easy-to-use tool, maybe the teams would be able to follow the RE process better, or at least the POs might be relieved a bit.

7.4 Future work

In this thesis, I have case studied the teams’ working situations, found out some problems, figured out some improvements, had one of the teams to try out the pilot and got their feedbacks on the pilot. There are some areas that could be further researched in the future.

As I have mentioned in section 1.4, this master thesis could have been cooperated more with the one which researched the testing work in the teams. Requirements are tightly related to testing in software development process. If there is a thesis that deals with these two aspects at the same time, some suggestion might be proposed from a more holistic viewpoint and the process could be improved to a higher level.

Also, if possible, the elicitation of requirements engineering of the teams can be investigated, so that more information of how the RE process is performed will be gained, and this is probably helpful for having more inspirations of generating the solutions to the RE problems in the teams.

8 Reference

[1] Agile 2008 Conference, program selection system, Starting a Kanban System for Software Engineering with Value Stream Maps and Theory of Constraints, http://submissions.agile2008.org/node/957 2009-06-21

[2] Agile Journal. The State of Scrum, http://www.agilejournal.com/index.php?option=com_content&task=view&id=886&Itemid=76 2009-06-12

[3] Agile Modeling. Agile Requirements Best Practices. http://www.agilemodeling.com/essays/agileRequirementsBestPractices.htm#SmallerIsBetter 2009-06-12

[4] Agile Modeling. Agile Requirements Modeling. http://www.agilemodeling.com/essays/agileRequirements.htm#Types 2009-06-12

[5] Agile Modeling. User Stories. http://www.agilemodeling.com/artifacts/userStory.htm 2009-06-21

[6] AppLabs, Testing in an Agile Environment, January, 2009, http://www.docstoc.com/docs/5852691/Testing-in-an-Agile-Environment 2009-06-12

[7] Aptoma, Estimation Using Planning Poker in Scrum. http://aptoma.com/select.star/2008/10/10/estimation-using-planning-poker/ 2009-06-21

[8] Architectural manifesto: Adopting agile development, Part 1. The meaning of Agile http://www.ibm.com/developerworks/architecture/library/ar-archman1/index.html 2009-06-12

[9] A.Cockburn and J.Highsmith. Agile Software Development. Addison-Wesley, 2002

[10] A. Eberlein and J. Leite. Agile Requirements Definition: A View from Requirements Engineering, In International Workshop on Time-Constrained Requirements Engineering, 9 September 2002

[11] A. Sillitti and G. Succi. Requirements Engineering for Agile Methods. In A. Aurum and C. Wohlin, editors, Engineering and Managing Software Requirements, page 315. Springer, 2005.

[12] B.Gillhan. Case Study Research Methods, Continuum, 2000.

[13] Developer Testing. The Developer Testing Paradox. http://www.developertesting.com/archives/month200501/20050127-TheDeveloperTestingParadox.html 2009-06-12

[14] D.Leffingwell. Mastering the Iteration: An Agile White Paper. Rally Software Development Corporation, 2007

[15]Functional and nonfunctional requirements. SearchSoftwareQuality.com. http://searchsoftwarequality.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid92_gci1288232,00.html, 2009-06-12

[16]F.Mårtensson and E.Karlsson. Test processes for a Scrum team, http://www.cs.lth.se/EDA920/presentations2009.shtml

[17]F. Paetsch, A. Eberlein and F. Maurer, Requirements Engineering and Agile Software Development, Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Page: 308, 2003

[18]G. Bergman, As a Gazelle to a Gazebo, Lean Magazine, Issue #4, January 2008

[19]G. Bergman, Lean reflections by an armchair general, Lean Magazine, Issue #4, January 2008

[20]I. Sommerville and P. Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons LTD, 1997

[21]I. Sommerville. Software Engineering - 7th Edition. Addison-Wesley, 2004.

[22]J. Itkonen, K. Rautiainen and C. Lassenius, Towards Understanding Quality Assurance in Agile Software Development, in International Journal of Agile Manufacturing, vol. 8(2), pp. 39-49, 2005.

[23]J. Tomyako, Engineering of Unstable Requirements Using Agile Methods, Carnegie Mellon University, August 2002

[24]K. Boness and R. Harrison, Goal Sketching: Towards Agile Requirements Engineering, Software Engineering Advances, 2007. ICSEA 2007. International Conference on Volume , Issue , 25-31 Aug. 2007 Page(s):71 – 71

[25]K. Schwaber, SCRUM Development Process, Advanced Development Methods, http://jeffsutherland.com/oopsla/schwapub.pdf 2009-06-12

[26]L. Cao and B. Ramesh, Agile Requirements Engineering Practices: An Empirical Study, IEEE Software, vol. 25, no. 1, pp. 60-67, Jan./Feb. 2008, doi:10.1109/MS.2008.1

[27]Manifesto for Agile Software Development. Principles behind the Agile Manifesto. http://agilemanifesto.org/principles.html 2009-06-12

[28]Maricopa Center for Learning & Instruction. Case Study Limitations http://www.mcli.dist.maricopa.edu/proj/res_meth/rmvl/case_limits.html, 2009-06-12

[29]M. Niazi and S. Shastry, Role of requirements engineering in software development process: an empirical study, Multi Topic Conference, 2003. INMIC 2003. 7th International, Volume , Issue , 8-9 Dec. 2003 Page(s): 402 - 407

[30]Mountain Goat, Planning Poker Cards, http://store.mountaingoatsoftware.com/ 2009-06-12

[31]M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley, 2004

[32]P.Abrahamsson, O. Salo, J Ronkainen and J Warsta. Agile Software Development Methods – Review and Analysis. VTT Electroniikka, 2002

[33]P.Duff, Case Study Research in Applied Linguistics, Lawrence Erlbaum Associates Inc, 2007

[34]R.Stake, The Art of Case Study Research, SAGE Publications, 1995

[35]R. Svensson, S. Birgisson, C. Hedin and B. Regnell, Agile Requirements Abstraction Model – Requirements Engineering in a Scrum Environment, Department of Computer Science, Lund University

[36]ScrumAlliance, Sprint Retrospective Meeting, http://www.scrumalliance.org/articles/39-glossary-of-scrum-terms#1113 2009-06-21

[37]Sony Ericsson Global, Profile, http://www.sonyericsson.com/cws/corporate/company/aboutus/profile 2009-06-12

[38]Succeeding with Agile. Non-functional Requirements as User Stories. http://blog.mountaingoatsoftware.com/non-functional-requirements-as-user-stories 2009-06-12 2009-06-12

[39]S. Bose, M. Kurhekar and J. Ghoshal, *Agile Methodology in Requirements Engineering*, Infosys Research Publication. http://www.infosys.com/research/publications/agile-requirements-engineering.pdf 2009-06-12

[40]Wikipedia. Agile Software Development. Contrasted with the water model. http://en.wikipedia.org/wiki/Agile_software_development#Contrasted_with_the_waterfall_model 2009-06-12

[41]Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, 29(4): 315-321.

Appendix

Interview questions to the product owners:

- How do you abstract requirements from the customers?
- Who interprets the use cases/user stories into tasks in backlogs? POs or Scrum Masters?
- If there are changes to the requirements, when do they happen usually? If everybody who will be affected by the changes show up? How to evaluate/investigate the changes to the requirements?
- Any problems/issues that you feel? Like the interaction with customers and the developers?
- How are the documents of the products created? What's the process of it (e.g. after discussing with whom, getting whose feedback, etc.)
- How often can you get access to the customers? For how long each time? Who will attend the requirements discussion, besides POs and customers? Will any relevant developers attend?
- When the demo is shown to customers, they find it is not what they want and the teams have to rework on it. So how to avoid it?

Interview questions to the developers (before the pilot):

- From your point of view, how is a product developed? Could you describe the process in brief?
- During your work, usually when do you need to look at the requirements?
- What do you think a "good" requirement should be?
- Usually during the meetings, can you understand the description of the tasks logged in Scrumwork well?
- Do you think usually you have got enough info in the requirements at the beginning of a sprint? If not, what kind of info do you think lacking, and what will you do to understand the requirements better?
- What else do you need on the basis of the current requirements?
- Have you ever misunderstood any requirement? Under which circumstance would the misunderstanding happen?
- Any challenges/barriers you have ever faced when working with requirements in an agile environment? What do you think has been good concerning the requirements?
- What do you think can be done to overcome these challenges/barriers?

- Have you ever over/under estimated any tasks? Under which circumstance would it happen?

- Do you think that the tasks have been prioritized when you have sprint planning meetings? How do you think tasks should be prioritized?

- How do you handle non-functional requirements such as performance or stability? How/from where do you get them?

- How do you generate use cases? How are test cases related to requirements?

- How are the requirements validated? (I.e., to make sure that a requirement is testable, traceable, etc.)

- Are tasks/stories changed throughout the development process? If so, how do you respond to this? Ex. Are the test documents updated?

- What do you think about the current product process in BRAT/QC in general, and about this interview?

Interview questions to the BRAT developers and the product owner (after the pilot):

- In general, how do you feel about the pilot sprint?

- What do you think about the changes? (User stories, done criteria, PO presentation, etc.) Is there anything you think especially good or not good enough?

- Is there anything can be improved or you want to have it improved? (The content or the way of doing it)

- Anything more you think can be done in the future sprints?

- Will you adopt the pilot items in the future?

- For the improvements that have not been tried in the pilot (e.g., non-functional requirements), do you think you will try them in the future?