FACULTY OF TECHNOLOGY

REQUIREMENTS ENGINEERING IN AGILE SOFTWARE PROJECTS

Eetu Rantanen

INDUSTRIAL ENGINEERING AND MANAGEMENT

Bachelor's Thesis

April 2017

FACULTY OF TECHNOLOGY

REQUIREMENTS ENGINEERING IN AGILE SOFTWARE PROJECTS

Eetu Rantanen

Supervisor: Kirsi Aaltonen

INDUSTRIAL ENGINEERING AND MANAGEMENT

Bachelor's Thesis

April 2017

TIIVISTELMÄ
OPINNÄYTETYÖSTÄ Oulun yliopisto Teknillinen tiedekunta

Koulutusohjelma (kandidaatintyö, diplomityö)	Pääaineopintojen ala (lisensiaatintyö)
Tuotantotalouden koulutusohjelma	
Tekijä Rantanen, Eetu	Työn ohjaaja yliopistolla Aaltonen K, apulaisprofessori

Työn nimi

Vaatimusmäärittely ketterissä ohjelmistoprojekteissa (Requirements engineering in agile software projects)

Opintosuunta	Työn laji	Aika	Sivumäärä
	Kandidaatintyö	Huhtikuu 2017	31 s.

Tiivistelmä

Monet ohjelmistoprojektit epäonnistuvat, koska tieto vaatimuksista on riittämätöntä toimituspäätöksiä tehdessä. Lisäksi projektinhallinnan prosessi, johon sisältyy ketterä vaatimustenhallinnan prosessi, on tunnistettu yhdeksi neljästä ketterien ohjelmistoprojektien menestystekijästä. Tämän takia ketterien ohjelmistoprojektien onnistumiseksi on tärkeää, että vaatimusmäärittelylle on selkeät ohjeet.

Tämän tutkimuksen tarkoituksena on analysoida ketterää vaatimusmäärittelyä ja löytää siinä yleisesti käytettyjä tapoja. Tavoitteena on määrittää jatkuva prosessi, jossa asiakkaan tarpeet tunnistetaan ja käännetään ohjelmiston vaatimuksiksi ketterässä ohjelmistokehityksessä. Tavoitteeseen pyritään tekemällä systemaattinen kirjallisuuskatsaus ketterään vaatimusmäärittelyyn. Ketterää ohjelmistokehitystä sekä perinteistä vaatimusmäärittelyä käsitellään muutaman perusteoksen pohjalta.

Tutkimuksen ylätason tutkimuskysymys on:

- Kuinka asiakkaan tarpeet käännetään vaatimuksiksi jatkuvana prosessina ketterissä ohjelmistoprojekteissa?

Lisäksi tutkimuksella on kaksi alatason tutkimuskysymystä:

1. Mitä asiakkaan tarpeet ovat ja kuinka ne tunnistetaan?
2. Minkälaisia tapoja ketterässä vaatimusmäärittelyssä käytetään?

Yleinen vaatimusmäärittelyprosessi sisältää neljä vaihetta. Ensin arvioidaan järjestelmän liiketoiminnallinen tarpeellisuus (kannattavuusselvitys). Tämän jälkeen etsitään vaatimuksia (selvitys ja analyysi) ja käännetään ne johonkin standardimuotoon (spesifikaatio). Viimeisessä vaiheessa tarkistetaan, että vaatimukset määrittävät järjestelmän juuri asiakkaan haluamalla tavalla (validointi).

Ketterässä vaatimusmäärittelyssä on neljä yleistä käytäntöä. Korkean tason kanssakäyminen asiakkaan ja kehitystiimin välillä, iteratiivinen eli toistava lähestymistapa vaatimusmäärittelyyn, vaatimusten priorisointi perustuen asiakkaalle syntyvään arvoon ja myös ei-funktionaalisten vaatimusten tunnistus. Lisäksi voidaan sanoa, että vaatimusten dokumentointi ketterissä menetelmissä on vähäistä.

Tämän tutkimuksen tuloksia voidaan yleisesti ottaen hyödyntää ja kehitettyä mallia voidaan käyttää vaatimusmäärittelyn ohjenuorana ketterissä ohjelmistoprojekteissa.

Muita tietoja

ABSTRACT
FOR THESIS

University of Oulu Faculty of Technology

Degree Programme (Bachelor's Thesis, Master's Thesis)	Major Subject (Licentiate Thesis)
Industrial Engineering and Management	
Author	**Thesis Supervisor**
Rantanen, Eetu	Aaltonen K, Assistant Professor

Title of Thesis

Requirements engineering in agile software projects

Major Subject	Type of Thesis	Submission Date	Number of Pages
	Bachelor's Thesis	April 2017	31 p.

Abstract

Many software projects are failed due to the delivery decisions that were made without adequate requirements information. In addition, the project management process including agile-oriented requirement management process has been identified as one of the four success factors in the agile software projects. Having the clear rules for requirements engineering is, therefore, an important thing for agile software projects from their success point of view.

In this study, the objective is to analyze agile requirements engineering and to find out practices that are used in it. The goal is to define a continuous process to identify customer needs and translate them into software requirements in the agile software development. This goal is going to be achieved by a systematic literature review on the agile requirements engineering. For the agile software development and the traditional requirements engineering, the theory has been gathered from some basic books of the theme.

The primary research question for this study is:

- How the customer needs will be translated into requirements in the agile software project as a continuous process?

There are also two secondary research questions:

1. What are the customer needs and how can they be identified?
2. What kind of practices are used in the agile requirements engineering?

Generally, the requirements engineering process includes four separate steps. First, the business usefulness of the system should be evaluated (feasibility study). After that, the requirements are discovered (elicitation and analysis) and converted into some standard form (specification). Last phase includes checking that the requirements define the system as customer wants (validation).

Agile requirements engineering includes four major practices. The high-level interaction between the development team and the customer, iterative approach for the requirements engineering, prioritizing the requirements based on their business value for the customer, and eliciting also the non-functional requirements. In addition, the documentation of requirements is minimalistic in agile approaches.

Results of this study can generally be applied and the model created can be utilized as a guideline when doing requirements engineering in the agile software projects.

Additional Information

TABLE OF CONTENTS

1 INTRODUCTION

The topic of this Bachelor's thesis is the requirements engineering in agile software development projects. Chow & Chao (2008) ranks the project management process as a success factor number four in the agile software projects. This factor includes also an agile-oriented requirement management process. In addition, many software projects are failed due to the delivery decisions that were made without adequate requirements information (Verner et al. 2008). Therefore, the topic of this thesis is important to the agile projects from their success point of view.

The goal of the thesis is to find and define a continuous process where the customer needs will be identified and translated into system requirements for the agile project team based on the previous research studies. The primary research question for this study is:

- How the customer needs will be translated into requirements in the agile software project as a continuous process?

There are also two secondary questions:

1. What are the customer needs and how can they be identified?
2. What kind of practices are used in the agile requirements engineering?

The goal mentioned above is going to be achieved by a systematic literature review on the topic of the thesis. For the agile software development and the traditional requirements engineering, the theory has been gathered from some basic books of the theme. A more systematic literature review has been done for the agile requirements engineering part. When searching the literature, the concepts 'requirements engineering' and 'agile' were used.

The structure of the thesis is as follows. The chapter two presents some basic values and principles of agile software development and introduces the role of the customer in the development process. The next chapter provides information about requirements engineering overall and the used methods in requirements engineering in agile

approaches. The researcher's own observations and views are discussed on the chapter four. After that, the last chapter provides a summary of the research and its results.

2 AGILE SOFTWARE DEVELOPMENT

In this chapter the concept of agile and the basic principles of agile software development are described. Also, at the end of the chapter, the question about the role of the customer in the agile software development will be discussed.

2.1 Concept of agile

Agile can be understood to be effective and steered easily to new directions. An agile process is both light and sufficient. The process must be light that it is possible to steer it easily to the new directions. The sufficiency of the process enables the staying in the game. The agility can be measured, for example, with the number of team members. A 40-person team will not be as agile as a six-person team. (Cockburn 2002, p. 178) Agility in its core means to be better able to promote quick response to changing environments, changes in user requirements, accelerated project deadlines, and such kind of things (Erickson et. al. 2005).

Williams & Cockburn (2003) describe that the basic software development methods such as, for example, plan-driven ones became unable to respond the shifting that both technology and business environment kept on during the project. In addition, the customers became increasingly unable to state their needs at the beginning of the project. In order to meet these requirements, Williams & Cockburn state that agility is about feedback and change.

2.2 Major values of Agile Software Development

Agile methods are an important approach to software engineering and the main idea of the agile methods is to maximize the business value of creating new software products. Agile methods are an efficient alternative to traditional systems engineering standards and document-driven development because of their customer-focused way to develop products. (Rico et al. 2009, p. 1)

Agile Manifesto was published by a group of software practitioners in 2001 (Beck et al. 2001). According to the manifesto, Agile Software Development has four major values:

Individuals and Interactions over processes and tools

This value means that developers are empowered to form self-managing teams. Also, the developers should be skilled and highly motivated computer programmers and they will work together to solve complex problems. (Rico et al. 2009, p. 9) In addition, business people and developers should work together daily during the project (Beck et al. 2001).

Working software over comprehensive documentation

The second value means that one of the highest priorities within agile methods is producing working software. Furthermore, this means that all resources are focused on producing the software in two- to four-week intervals. This is the way to maximize the business value because the customer is paying for the working software, not for the documentation. (Rico et al. 2009, p. 9) A working software is also a primary measure of progress done by development team (Beck et al. 2001). Abrahamsson et al. (2002) points out that the developers are supposed to keep the code as simple as possible that the documentation burden can be decreased to an appropriate level.

Customer collaboration over contract negotiation

Customer collaboration is one of the key point of the agile software development (Paetsch et al. 2003). In a sense, it means that developers ask the customers what they want. In addition to asking, listening to and interacting with customers are also ways to ascertain their needs. (Rico et al. 2009, p. 8)

Responding to change over following a plan

Agile methods welcome changes in requirements even late in the development (Beck et al. 2001). This means that development process has to be planned with minimum effort and repeated often until the customer is completely satisfied. Customer needs evolve and change during the project and planning iteration without huge investments in documentation opens a clear path to the best business value. (Rico et al. 2009, p. 10)

2.3 Basic principles of Agile Software Development

According to Cockburn (2002, p. 178-180), part of getting to agile is that the 'sweet spots' of effective software development are identified and the project is moved as close as possible to those 'sweet spots' or principles. Below are listed the five principles that have been proposed by Cockburn (2002):

Two to Eight People in One Room

This principle is for the communication. When a small project team is sitting in the same room they are able to ask small questions and the questions can be answered very quickly. Also, the design ideas and project plan is kept on the board and everyone can easily take a look at it.

Onsite Usage Experts

It is very useful to have a usage expert available at all time because the feedback time is then as short as possible. The development team can gain a deeper understanding of the needs and habits of the users. Also, developers are able to test more ideas and by that way increase the quality of the final product.

One-Month Increments

This principle is also needed for ensuring the rapid feedback. Now, the feedback concentrates both on the product and the development process itself. Incremental development is perfect for providing feedback points which allows the requirements and the process being repaired quickly. The biggest question is that how long the delivery increments should be. There is not only one right answer to this question but people seem to be able to best focus on their effort for about three months.

Fully Automated Regression Tests

Fully automated regression tests can be either unit or functional tests or both. Using automated regression tests have two main advantages. First, the code can be freely modified or entire modules can be improved or replaced because the automated tests will

uncover possible bugs. Second, the developers themselves can enjoy better quality of life because they do not need to think about the possibility that someone has broken their code.

Experienced Developers

In an ideal situation, the development team consists of only experienced developers because they can be many times as effective as inexperienced ones. When the team consists of only experienced developers the number of team members can be dramatically decreased. This is, of course, hard to land and in case that there are inexperienced developers in the team it can be helpful to bringing a trainer or mentor for them.

Abrahamsson et al. (2002) provides a good summary of what makes a development method an agile one. In their opinion, the software development is agile when the development is incremental, cooperative, straightforward, and adaptive.

2.4 Scrum as an example

There are many types of agile methods. Today's five major agile methods are Crystal Methods, Scrum, Dynamic Systems Development, Feature Driven Development, and Extreme Programming, amongst which Scrum and Extreme Programming are the most used methods. (Rico et al. 2009, p. 25) Scrum is based on an empirical process control model instead of the traditional defined process control model. Because of that difference, Scrum works well in the context of systems development projects. (Schwaber & Beedle 2002, p. 89-104)

The Scrum as a framework consists of Scrum Teams and their associated roles such as Scrum Master and Product Owner, events, artifacts, and rules. The Scrum will not success and it cannot be used without any component within the framework. The events in the Scrum are used to create regularity and to minimize the need of unnecessary meetings. These events are Sprint, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective. The artifacts in the Scrum represent work or value and they are designed to maximize the key information. By that way everyone has the same understanding of

an artifact. These artifacts are Product Backlog and Sprint Backlog. (Schwaber & Sutherland 2013)

The abovementioned events and artifacts are shown in the basic model of a Scrum development process in the figure 1.

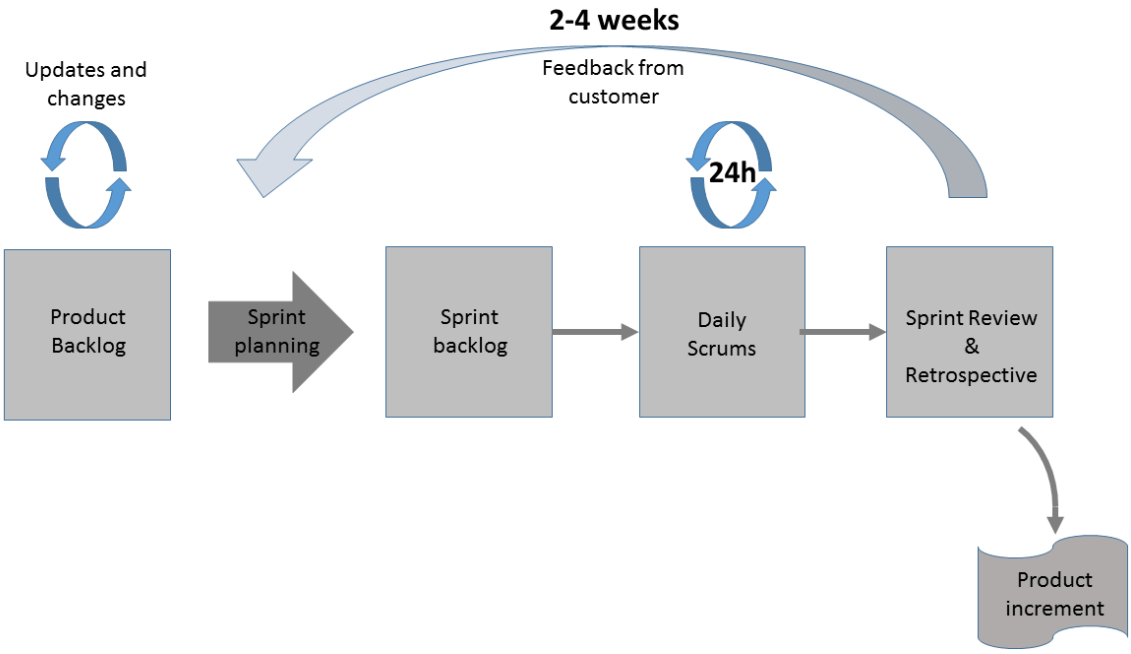

Figure 1. Typical model of a Scrum development process (According to Schwaber & Sutherland 2013)

2.5 The role of the customer in Agile Software Development

The customer has a very essential role in Agile Software Development. Customer satisfaction, customer collaboration, and customer commitment are three of the nine founded success factors in the agile software projects. Customer commitment has also a strong relationship with success. (Misra et al. 2009) In addition, customer collaboration is one of the four major values of Agile Software Development as described in the chapter 2.2.

Participants in the workshop of Customer Involvement in Extreme Programming in 2001 (van Deursen 2001) listed the following tasks for a customer:

1. To understand customer wishes, to maintain regular contact with end users, and to balance end users' potentially conflicting interests.
2. To talk to developers, clarify feature requests when needed, and understand some of the technical concerns developers have.
3. To specify functional tests for user stories (requirements), and verify that these tests run correctly.
4. To participate in the planning of iterations and releases.
5. To maintain good contact with management, explain progress, and justify the time that is spent with the development team.

Sillitti & Succi (2005) points out that in the customer on-site practice the customer must have couple of attributes. First of all, customer should be available and answer the questions coming from the development team. Delays in the answers delay the project. The customer should also be a domain expert and be able to answer every question. He or she knows how the application should work. Lastly, the customer should be able to make final decisions and commitments. This allows a fast decision making process for requirements changes, for example. (Sillitti & Succi 2005) Martin et al. (2004) founded that customers were consistently under significantly more pressure than the developers or other participants in the project.

3 AGILE REQUIREMENTS ENGINEERING PRACTICES

This chapter starts with a brief presentation about the requirements engineering in the software project and provides also a model for traditional requirements engineering process. After that the practices of the agile requirements engineering are discussed in more detail.

3.1 Traditional requirements engineering

Requirements engineering provides the mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a solution, specifying the solution, validating the specification, and managing the requirements. The goal of requirements engineering is to ensure that the system have been specified to meet the customer's needs and to satisfy the customer's expectations. (Pressman 2000, p. 251-252)

The overall process for traditional requirements engineering includes four high-level sub-processes. These are concerned with evaluating the business usefulness of the system (feasibility study), discovering requirements (elicitation and analysis), converting requirements into some standard form (specification), and checking that the requirements define the system as customer wants (validation). (Sommerville 2004, p. 143) In addition, requirements must be managed because they will change throughout the life of the system (Pressman 2000, p. 256).

Feasibility study

For all new systems, the requirements engineering process should start with a feasibility study. The idea of a feasibility study is to determine whether it is worth continuing with the requirements engineering and system development process or not. When carrying out a feasibility study, an information assessment, information collection, and report writing are involved. Below are some examples of possible questions to be put in the context of information collection and assessment. (Sommerville 2004, p. 144-145)

1. How would the organization cope if this system were not implemented?

2. What are the problems with current processes and how would a new system help alleviate these problems?
3. What direct contribution will the system make to the business objectives and requirements?
4. Can information be transferred to and from other organizational systems?
5. Does the system require technology that has not previously been used in the organization?
6. What must be supported by the system and what need not be supported?

Once the information has been collected, the feasibility report will be written. In the report, there should be a recommendation whether the system development should continue or not. (Sommerville 2004, p. 146)

Requirements elicitation and analysis

In this activity, the software engineers work with the customers and end-users to find out the application domain, services that system should provide, the required performance of the system, and so on (Sommerville 2004, p. 146). There are some identified problems that make the requirements elicitation difficult. To overcome these problems, the requirements gathering activity should be approached in an organized manner. (Pressman 2000, p. 252) The overall process would ideally be following: (1) elicit the overall goals for the system, (2) elicit the information about present work and present problems, (3) elicit the detailed issues that the system shall deal with, (4) look at the possible solutions, and (5) transform the issues and possibilities into requirements. (Lauesen 2002, p. 331)

There are different kind of activities for eliciting and analyzing the requirements. For example, interviewing, scenarios, and use-cases can be used in the interaction with the customer and other stakeholders. (Sommerville 2004, p. 146-156) Interviewing is a good method for getting knowledge about the present work and the present problems in the domain. It can also give information about what is realistic and where the conflicts may lie but other techniques are needed for verifying the information. (Lauesen 2002, p. 339) Lauesen (2002, p. 338-350) presents also a big number of other techniques for requirements elicitation such as observation, questionnaires, brainstorming, focus groups, prototyping, and negotiating.

After that requirements are gathered, the requirements analysis will follow. In this phase the requirements are categorized and organized into related subsets, each requirement is explored in relationship to others, the consistency of the requirements is examined, and the requirements are prioritized based on the customer needs. The customers, users, and other stakeholders are asked to prioritize the requirements and then discuss about the conflicts in priorities. An important thing is also to identify and analyze all risks associated with each requirement. (Pressman 2000, p. 253-254)

Requirements specification

The requirements specification means different things to different people when it is discussed in the context of software. For some people specification means a written document while for some people it can be a graphical model or a collection of usage scenarios. According to Lauesen (2002, p. 376-379), a good requirements specification fulfills eight criteria of quality. That criteria says that a good requirement specification is correct, complete, unambiguous, consistent, ranked for importance and stability, modifiable, verifiable, and traceable. (Lauesen 2002, p. 376-379) However, it is sometimes necessary to remain flexible when developing a requirements specification. The type of the specification may depend on the size of the software to be developed. For example, for large systems, a written document with natural language descriptions and graphical models may be the best approach but for the smaller products the usage scenarios may be all that are required. (Pressman 2000, p. 254)

Requirements validation

The mean of the requirements validation is to show that the requirements define the system as the customer wants. Requirements validation checks the requirements for validity, consistency, completeness, realism, and verifiability. (Sommerville 2004, p. 158-160) Pressman (2000, p. 255) points out that this is called the quality of the requirements. In addition, it is important to validate that stakeholders are happy and think about the major risks (Lauesen 2002, p. 373).

The most common techniques that are used in the requirements validation are requirements reviews and prototyping (Sommerville 2004, p. 158-160). In the reviews,

the requirements specification is looked carefully by developers, customer, and analyst (Lauesen 2002, p. 390-393). Also, Pressman (2000) presents reviews as a primary mechanism for requirements validation. In his approach, which is the formal technical review, also the other stakeholders are included in the review team. (Pressman 2000, p. 255) During the review meeting, reviewers report the problems or defects they observed but they are not allowed to discuss about the possible solutions. Prototype tests are also good because they enable seeing whether the requirements are meaningful and realistic. (Lauesen 2002, p. 390-393)

Requirements management

Requirements management is needed because business, organizational, and technical changes inevitably lead to changes in requirements (Sommerville 2004, p. 161). During development, some requirements turn out to be wrong, new demands are discovered, some requirements are too expensive to meet, or the parties agree to implement something simpler than originally described (Lauesen 2002, p. 322). Requirements management is a process to manage and control these changes. When there is a proposed change in a requirement, it should be analyzed and the impact of it should be assessed. (Sommerville 2004, p. 161-166) Pressman (2000) points also out how important the traceability of the requirements is for the requirements' management activities. If the system is very large, it is important to track the changing requirements efficiently. (Pressman 2000, p. 256)

Lauesen (2002) presents a basic chain for the change cycle. First, the requirement issue is reported to the change control board. Second step is to analyze the issue together with other issues. Is it a new demand, a request to change a requirement, or a misunderstanding? Also, the costs and benefits of the change issue need to be analyzed so that the decision of the issue can be made. The next step is then to make a decision whether the request about requirement should be rejected or included. Fourth step is to reply and inform the decision to the source of request and other people impacted by it. Finally, the decision must be carried out. (Lauesen 2002, p. 322-323)

3.2 Agile requirements engineering practices

3.2.1 Interaction between the development team and the customer

To have a customer accessible or on-site is a key point in all agile approaches (Paetsch et al. 2003). The overall role and attributes of the customer in agile software development are discussed in the chapter 2.5. Paetsch et al. (2003) points also out that the customer is involved under whole the development time but it is not guaranteed that every user or customer from necessary backgrounds are present. In agile software development, it is highly recommended to have no communication layers between the development team and the customer. When the development team communicates directly with the customer, the probability of misunderstandings between parties reduces significantly. (Sillitti & Succi 2005)

The aim of agile requirements engineering is to effectively translate ideas from the customer to the software development team, rather than create extensive documentation. When using informal communication between the customer and the development team, the time-consuming documentation and approval processes are not needed anymore. These things are perceived as unnecessary when the requirements are evolving. Most organizations use simple techniques such as user stories to define requirements from customer. (Cao & Ramesh 2008) When collecting the requirements from customer the whole development team should be involved and the common language of customer should be used. This will reduce the probability of misunderstandings. In addition, if the requirements are too complex, the customer is asked to split them in the simpler ones. (Sillitti & Succi 2005)

Continuous interaction between the customer and the development team has some benefits. First, the customers can steer the project to the directions that were unanticipated. Especially, when the requirements are evolving due to changes in the environment or customer's understanding of the solution. (Cao & Ramesh 2008) Second, direct interaction with customer does not have the same problems of misunderstandings as having long chains of knowledge transfers has. This was discussed at the beginning of this chapter. Also, talking directly with the customer helps establishing trust relationships. (Paetsch et al. 2003)

On-site customer representation is difficult in most cases. If there is no possibility to an intensive interaction between customer and developers, this approach may result inadequately developed or totally wrong requirements. The customer may have more than one group involved to the project and each of them having interest in different aspects of the system. In this kind of case, it is hard to achieve a consensus between customer groups and then the project team must spend extra effort to integrate the requirements. (Cao & Ramesh 2008) Sillitti & Succi (2005) recommend that in the projects, that have a limited size, the problem of having more than one group involved to the project is solved by reducing the number of groups or persons to one. The person who represents all the stakeholders should be a domain expert and be able to make important decisions such as prioritize requirements. (Sillitti & Succi 2005)

3.2.2 Iterative requirements engineering

In the agile methods, functionalities are released in small and frequent cycles. This allows the development team to get more and real-time feedback from customer. (Sillitti & Succi 2005) At the beginning of the project the development team acquires a high-level understanding of the software's critical features. Reasons for not to set too much effort on requirements engineering at the beginning include high requirements volatility, incomplete knowledge of technology used or customers who cannot clearly describe the requirements before they see them. After the brief requirements identifying at the beginning, agile requirements engineering continues at each development cycle. At the start of each cycle, the customer and development team meets and they discuss about the features that must be implemented. (Cao & Ramesh 2008) By that way the requirements are detailed gradually and iteratively as the development progress (Bjarnason et al. 2011).

Gradual detailing ensures that requirements are actively worked with throughout the development process, which reduces the challenge of communication gaps within development as well as between business and development team (Bjarnason et al. 2011). An iterative approach for requirements engineering also makes it easier to keep system requirements specification up to date and creates more satisfactory relationship with the customer (Cao & Ramesh 2008, Bjarnason et al. 2011). When the relationship with the customer is good, the customer will provide feedback for the development team. Iterative requirements engineering enables getting frequent feedback from customer. This helps reducing the waste in requirements such as unnecessary features. (Sillitti & Succi 2005)

According to the Cao & Ramesh (2008) the iterative requirements engineering can also be used in the stable environments where the changes in the requirements come from unforeseen technical issues.

Iterative requirements engineering faces at least two major challenges. Cost and schedule estimation for the project are difficult to do at the beginning because the project scope is subject to constant change. Obtaining management support for such projects can be challenging. (Cao & Ramesh 2008) Second, the lack of clear requirements picture at the beginning of the development will result in significant changes in requirements during the development. However, this approach will easier lead to a more feasible scope. (Bjarnason et al. 2011)

3.2.3 Requirements prioritization

In the agile development, the highest-priority features are implemented early so that the customers can realize the most business value (Paetsch et al. 2003, Cao & Ramesh 2008). The prioritization should be repeated frequently during the whole development process because the understanding of the project increases and new requirements are added during development (Paetsch et al. 2003). Cao & Ramesh (2008) and Sillitti & Succi (2005) recommends that the requirements will be prioritized at the beginning of each development cycle. The customer and the development team assign priorities for each feature to be secure that those requirements that are most important will be implemented first (Sillitti & Succi 2005). Cao & Ramesh (2008) says also that the requirements prioritization is based only on one factor, the business value for customer.

Sillitti & Succi (2005) presents a four-step process for requirements prioritization:

1. The development team estimates the time required to implement the functionality.
2. The customer sets business priorities for each functionality.
3. The development team assigns a risk factor for each functionality according to the business priorities.
4. The features to be implemented in the iteration are identified by the development team and the customer.

When repeating the requirements elicitation and these four steps of prioritization at the beginning of each iteration, it is possible to identify the requirements that will not provide enough value for customer. (Sillitti & Succi 2005)

When customers are involved in development process, they can provide business reasons for each requirement at any development cycle. By this way, the development team will achieve a clear understanding of the customer's priorities and they can better meet the customer needs. (Cao & Ramesh 2008) Prioritizing the requirements also reduces the possibility of including waste in requirements. This means such requirements that will not provide a real benefit for customer's business. (Sillitti & Succi 2005) However, if business value is the only or primary criterion for requirements prioritization, it might result major problems as non-scalable architecture. Also, it might result a system that cannot accommodate requirements that appear secondary to customer but that become critical for operational success. (Cao & Ramesh 2008) These non-functional requirements are discussed in the next chapter. Cao & Ramesh (2008) remind also that the continuous reprioritization may lead to instability if it is not practiced with care.

3.2.4 Non-functional requirements

Non-functional requirements can be understood as the constraints under which the entire system must operate. In other words, the non-functional requirements do not tell what the software will do but how the software will do it. (Adams 2015) Non-functional requirements are, for example, scalability, maintainability, portability, safety, or performance (Cao & Ramesh 2008).

In the agile approaches, the non-functional requirements are often neglected. Customers often focus more on the core functionality. (Cao & Ramesh 2008) The customers are telling what they want the system to do and, therefore, they do not normally think about non-functional requirements (Paetsch et al. 2003). One common exception is, however, that customers focus on some requirements of the ease of use, interface, and safety (Paetsch et al. 2003, Cao & Ramesh 2008). Because of that low impact for non-functional requirements from the customer's side, the development team should guide the customer so that such hidden needs can be identified (Sillitti & Succi 2005). De Lucia & Qusef (2010) propose also that the customers and agile team leaders would arrange for meetings to discuss non-functional requirements even in the earliest stages.

Sillitti & Succi (2005) points also out that the need of specifying non-functional requirements is not so important in the context of agile software development than in the other context because of the continuous interaction with the customer. Customer can test software after each iteration and if he identifies some problems regarding non-functional requirements, the development team can adapt the system to meet such requirements in the following iteration. (Sillitti & Succi 2005) It is important for the development team to know most of the non-functional requirements because they can affect the choice of the architecture (Paetsch et al. 2003). Considering non-functional requirements by this way may have a big risk because there is a lack of specific techniques to manage them (Sillitti & Succi 2005).

3.3 Documentation in agile requirements engineering

In the agile practices the documentation is minimalistic and the requirements are not always documented (Bjarnason et al. 2011). However, some agile methods recommend to use requirements document but its extend depends on the development teams' decision. Also, the team size should be considered when planning the documentation. The agile team is considered to be more cost-effective and productive when overdocumenting is avoided. The minimal documentation also increases the chances to keep the document up to date when the software is changed. (Paetsch et al. 2003)

Customers often ask the team to produce documentation before the team is resolved but the scope of this documentation is very limited and concentrated on the core aspects of the system. Although the modeling is used as a part of the agile requirements engineering, most of the models will not become part of the persistent documentation of the system. (Paetsch et al. 2003)

3.3.1 User stories

A user story describes functionality that a user or a customer of the software or system keeps valuable. There are three aspects that the user stories are composed of:

- A written description of the story
- Conversations about the story

- Tests that convey and document details

While the written description may contain the text of the story, the detailed information is worked out in the conversations and is documented in the tests. (Cohn 2004, p. 4) The written description can concern a small piece of functionality and be written on a piece of paper. These papers are then hanged on a pin board. (Sillitti & Succi 2005) One user story does not contain very many details and, therefore, a basic question is that where the details are. Cohn (2004) answers this question and says that many of the requirement details can be expressed as additional stories. It is better to have more stories than have some stories that are too large. (Cohn 2004, p. 5)

3.3.2 Challenges of minimal documentation

In an overall look, agile methods tend to err on the side of producing not enough documentation. The lack of documentation might cause problems in agile teams because the documentation is used to share information between people. (Paetsch et al. 2003) A communication breakdown can occur owing to personnel turnover, rapid changes to requirements, unavailability of the right customer representative, or the application's growing complexity (Cao & Ramesh 2008).

In the case of communication breakdown, there can be many problems. Problems are for example inability to scale the software, evolve software over the time or take new members into the development team. (Cao & Ramesh 2008) A new team member will have many questions regarding the project and if he or she needs to ask other members whole the time, it will slow down the work (Paetsch et al. 2003).

4 DISCUSSION

In the previous chapters, there have been discussion about agile software development, traditional requirements engineering process, and the practices that have been used in agile requirements engineering. This chapter connects these pieces of information together and provides a model for requirements engineering in the agile software projects. In addition, the constraints of the study are discussed in the end of this chapter.

4.1 A model for agile requirements engineering

As we can summarize from the chapter three, the requirements engineering in agile software development has four common practices. These practices are customer collaboration, iterative requirements engineering, requirements prioritization, and non-functional requirements. To use each of these in one project can be easily justified.

Customer collaboration is one of the four major values of agile software development. It plays an important role especially in the requirements engineering because it is the customer whose needs the software should meet. In the chapter 2.5, the role of the customer in the agile software development have been clarified. That chapter shows that some of the main things that customer should do in the agile software development is to talk to the developers and clarify feature requests when needed. Continuous interaction between the developers and the customer seems to reduce the need of the time-consuming documentation as have been discussed in the chapter 3.2.1.

Agile software development can be recognized as an iterative activity. A working piece of software is delivered to the customer frequently. When considering this, it seems to be obvious and very useful that the requirements engineering is also done iteratively. This does not mean that every phase of traditional requirements engineering process (3.1) should be repeated in each iteration. However, those phases that help developers to get better understanding of customer's needs and requirements are good to be repeated in each iteration. Those are requirements elicitation and analysis, requirements specification, and requirements validation. In the agile software development, though, the requirements specification does not mean any extensive documentation as has been pointed out in the chapter 3.3.

While agile software development concentrates on delivering most business value to the customer, the requirements prioritization is one of the key things in agile requirements engineering. Requirements should be prioritized continuously in order to response evolving needs of the customer. As have been explained in the chapter 3.1, the requirements prioritization occurs in the requirements analysis phase. In the context of agile software development, we should, however, notice that the prioritization is based primarily on the business value for customer.

The result from the chapter 3.2.4 was that agile methods do not consider non-functional requirements too much. They are often neglected. Although there is not an efficient process to elicit, analyze, and manage the non-functional requirements, it is important for developers to understand the fact that customers will seldom think about them.

Understanding these practices will give a base knowledge to do requirements engineering in agile software projects efficiently. However, in my opinion, it would be much easier and clearer to have a model for agile requirements engineering. By taking abovementioned practices and the characteristics of agile software development into consideration, I have created a model for agile requirements engineering. The model itself is presented in the figure 2. The figure 3 shows how the agile requirements engineering fits in the scrum development process presented in the figure 1.

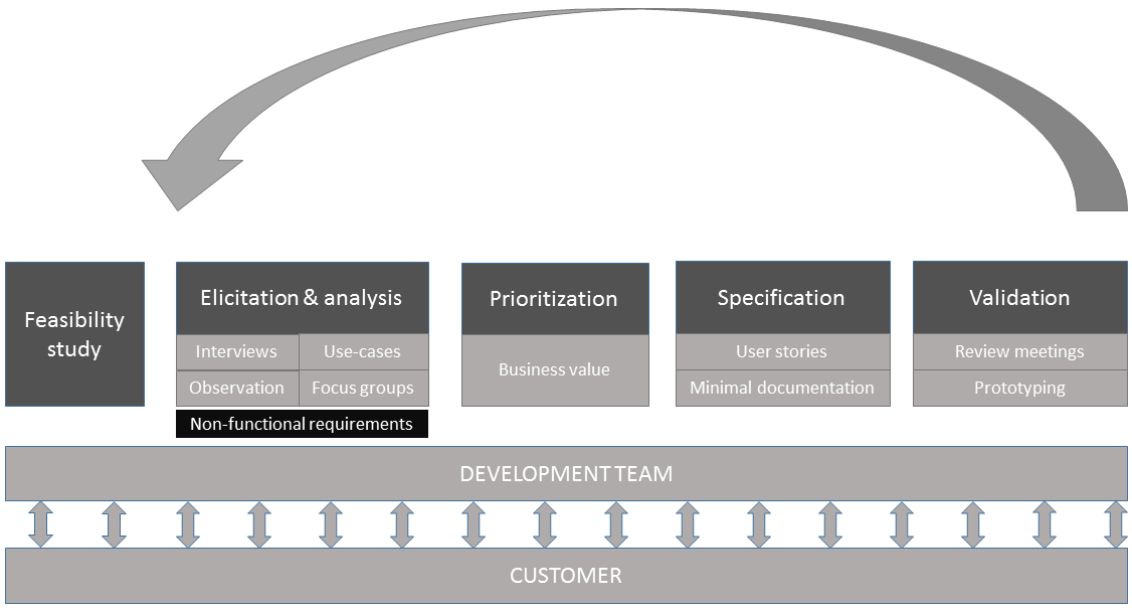

Figure 2. A model for agile requirements engineering

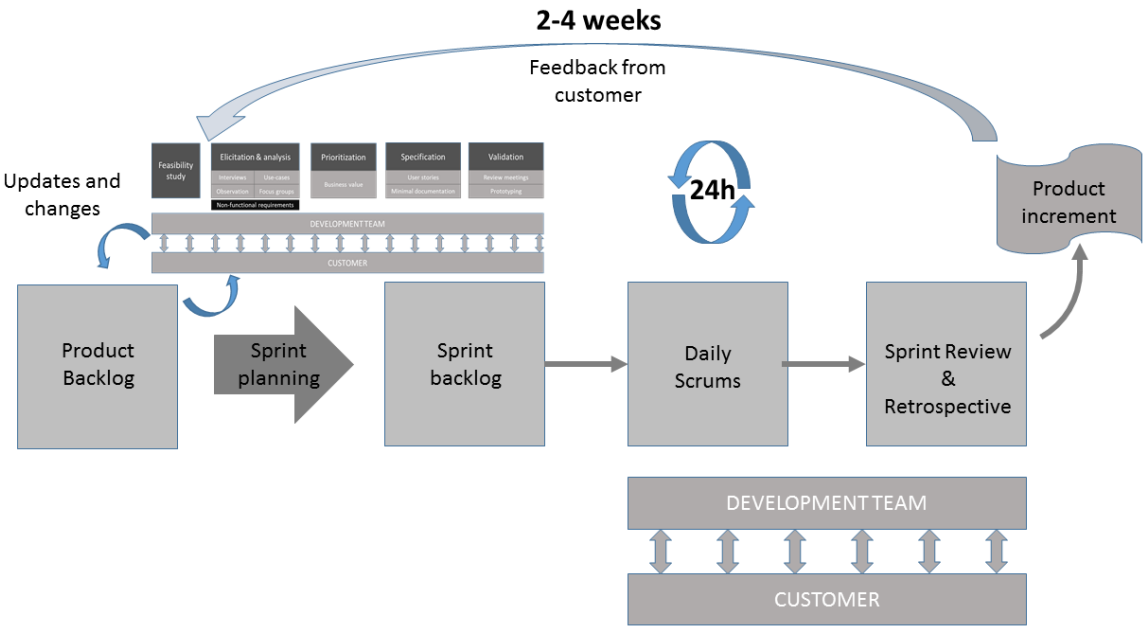

Figure 3. Agile requirements engineering in the typical model of scrum development process

4.2 Evaluation of the model

As have been said, the model of agile requirements engineering can be used in the context of agile software projects. Therefore, in my opinion, this model can be useful for companies whose business is based on the software projects. In the introduction of this study, there were named a couple of reasons why requirements engineering is an important part of the software projects. I think that the model presented above will lead to more successful projects because the practices used in it have been identified by the empirical studies.

However, the model also has some drawbacks. In the model, the requirements engineering occurs only at the beginning of each iteration. What happens if the customer needs change during the iteration? The model does not guide how the changing needs or requirements should be handled in that case. In addition, the figure 3 does not show the right order of requirements engineering and sprint planning. Phase sprint planning means that the most

important requirements are chosen to be done in the next iteration. The model does not consider that requirements prioritization and sprint planning can occur concurrent.

4.3 Constraints of the study

This study was performed by a literature review. Literature review succeeded and the information needed were found. However, in this kind of topic, the empirical study or a case study could have provided better results. We would have, for example, been able to ask organizations whether they have to do requirements engineering during whole the iteration or is it adequate to do it only at the beginning. According to answers to this question the model could have been changed.

Also, the other constraint was the length of the report to be written. Bachelor's Thesis is not a very large report and this limited a bit, for example, the number of the theory chapters and how thoroughly the theory was discussed. However, in my opinion, regardless of these constraints, the research questions were well answered in the study.

5 CONCLUSION

This study aimed to find and define a continuous process where the customer needs are identified and translated into system requirements in the context of agile software development. The continuous process was tried to be defined based on the information from basic books of agile software development and previous researches on the topic.

The first theory chapter presented the basic values and principles of the agile software development. As a summary of the agile methods, we can state that they are commonly used in the quickly changing environments such as software projects. Agile methods take the changes in consideration by being iterative, incremental, and cooperative.

After that, we looked at the requirements engineering and discussed about the practices that are used in the agile requirements engineering. The agile requirements engineering practices were identified by studying previous researches on the topic. As a result, four major practices were found. First, the high-level interaction between the development team and the customer was stated as the most important thing. In addition, requirements engineering should be iterative and the requirements should be prioritized based on their business value for the customer. Finally, the non-functional requirements should be considered when eliciting the requirements. Also, the documentation of requirements was founded to be minimalistic in agile approaches.

In the discussion chapter, the model of agile requirements engineering was presented. The model considered the practices that were identified in the previous chapter as well as the basic characteristics of agile software development. In addition, the model was evaluated and the constraints of the study was discussed.

The subject of agile software development is nowadays researched quite well. However, the process of eliciting and managing the non-functional requirements is not very clear in the agile approaches. As can be read from the chapter 3.2.4, the non-functional requirements are often neglected in the agile development projects. Therefore, it would be helpful to direct the future research on this topic to the area of non-functional requirements and try to identify an efficient and well-organized process to elicit and manage them.

REFERENCES

Abrahamsson P., Salo O., Ronkainen J. & Warsta J., 2002. Agile Software Development Methods: Review and Analysis. VTT Publications 478. VTT, Espoo. Available at: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf, [Referenced 24.3.2017]

Adams K.M., 2015. Non-functional Requirements in Systems Analysis and Design. Switzerland: Springer International Publishing, 264 p. ISBN 978-3-319-18344-2

Beck K. et al., 2001. Agile Manifesto. Available at: http://agilemanifesto.org, [Referenced 24.3.2017]

Bjarnason E., Wnuk K. & Regnell B., 2011. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. Proceedings of the 1st Agile Requirements Engineering Workshop, AREW'11 - In Conjunction with ECOOP'11.

Chow T. & Cao D., 2008. A survey study of critical success factors in agile software projects. The Journal of Systems & Software, vol. 81, no. 6, pp. 961-971.

Cockburn A., 2002. Agile Software Development. 2nd edition. Boston: Pearson Education, Inc., 278 p. ISBN 0-201-69969-9

Cohn M., 2004. User Stories Applied – For Agile Software Development. Boston: Pearson Education, Inc., 268 p. ISBN 0-321-20568-5

De Lucia A. & Qusef A., 2010. Requirements engineering in agile software development. Journal of Emerging Technologies in Web Intelligence, vol. 2, no. 3, pp. 212-220.

Erickson J., Lyytinen K. & Siau K., 2005. Agile modeling, agile software development, and extreme programming: The state of research. Journal of Database Management, vol. 16, no. 4, pp. 88.

Lan Cao & Ramesh B., 2008. Agile Requirements Engineering Practices: An Empirical Study. IEEE Software, vol. 25, no. 1, pp. 60-67.

Lauesen S., 2002. Software Requirements – Styles and Techniques. England: Pearson Education Limited, 591 p. ISBN 978-0-201-74570-2

Martin A., Biddle R. & Noble J., 2004. The XP customer role in practice: three studies. Agile Development Conference, IEEE, pp. 42-54.

Misra S.C., Kumar V. & Kumar U., 2009. Identifying some important success factors in adopting agile software development practices. Journal of Systems and Software, vol. 82, no. 11, pp. 1869-1890.

Paetsch F., Eberlein A. & Maurer F., 2003. Requirements engineering and agile software development. Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, pp. 308.

Pressman R.P., 2000. Software Engineering – A Practitioner's Approach, 5th edition. England: McGraw-Hill Publishing Company, 915 p. ISBN 0-07-709677-0

Rico D.F., Sayani H.H., & Sone S., 2009. The business value of agile software methods. U.S.A.: J.Ross Publishing, Inc., 214 p. ISBN 978-1-60427-031-0

Schwaber K. & Beedle M., 2002. Agile Software Development with Scrum. United States of America: Prentice-Hall Inc., 158 p. ISBN 0-13-207489-3

Schwaber K. & Sutherland J., 2013. The Scrum Guide™ – The Definitive Guide to Scrum: The Rules of the Game. Available at: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf, [Referenced 23.3.2017]

Sillitti A. & Succi G., 2005. Requirements engineering for agile methods. Engineering and Managing Software Requirements, pp. 309-326.

Sommerville I., 2004. Software Engineering, 7th edition. England: Pearson Education Limited, 759 p. ISBN 0-321-21026-3

van Deursen A., 2001. Customer involvement in extreme programming. ACM SIGSOFT Software Engineering Notes, vol. 26, no. 6, pp. 70-73.

Verner J., Sampson J. & Cerpa N., 2008. What factors lead to software project failure? Research Challenges in Information Science, IEEE, pp. 71-80.

Williams L. & Cockburn A., 2003. Agile software development: it's about feedback and change. Computer, vol. 36, no. 6, pp. 39-43.