OULUN AMMATTIKORKEAKOULU

Antti Pekkala

INTRODUCING AGILE WORK CULTURE TO WATERFALL-BASED SOFTWARE DEVELOPMENT ORGANIZATION

INTRODUCING AGILE WORK CULTURE TO WATERFALL-BASED SOFTWARE DEVELOPMENT ORGANIZATION

Antti Pekkala
Master's thesis
Fall 2017
Technology Business
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme, Technology Business

Author(s): Antti Pekkala
Title of thesis: Introducing Agile Work Culture to Waterfall-Based Software Development Organization
Supervisor(s): Lasse Haverinen
Term and year when the thesis was submitted: Fall 2017 Number of pages: 53

The objective of this thesis was to recognize the most required changes and new practices when the target organization is moving towards from Waterfall-based approach to Agile and Lean practices, and planning to take Scrum process into use. The thesis focuses on R&D Development Teams and Product Management of the target organization.

The knowledge-base of the thesis focuses on presenting theories behind the most commonly known software development processes. The oldest process known as Waterfall model is described, and after that Agile and Lean theories are explained more deeply, as well as modern software development processes based on these. The theoretical part of the thesis also provides reasons why these more modern processes are better than the Waterfall model.

The result of the thesis provides several different required changes for the target organization required to be done. These are related to changes on people behaviours and skills, and role changes. The results also describe answers to a few practices when Scrum is taken into use.

Keywords: Agile, Lean, Scrum, Waterfall mode, software development

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tutkinto-ohjelma, Suuntautumisvaihtoehto

Tekijä(t): Antti Pekkala
Opinnäytetyön nimi: Introducing Agile Work Culture to Waterfall-Based Software Development Organization
Työn ohjaaja: Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 53

Työn tarkoituksena oli tunnistaa eniten vaaditut muutokset ja uudet käytännöt, kun työn kohdeorganisaatio on siirtymässä Vesiputousmalli pohjaisesta lähestymistavasta kohti Agile ja Lean käytäntöjä, ja on suunnittelemassa ottamaan käyttöön Scrum prosessimallin. Työ keskittyy kohdeorganisaation R&D:n kehitystiimeihin ja tuotehallintaan.

Työn tietopohja keskittyy esittämään teoriaa yleisesti tunnetuista ohjelmistokehityksessä käytetyistä prosesseista. Työssä on kuvattu vanhimpana ohjelmistokehitys prosessina tunnettu Vesiputousmalli ja sitten syvennytty Agile:n ja Lean:in teorioihin, ja näihin pohjautuviin moderneimpiin ohjelmistokehityksen prosesseihin. Työn tietopohja antaa myös perusteita miksi nämä modernimmat prosessit ovat parempia kuin Vesiputousmalli.

Työn tuloksena on saatu useita tarvittavia muutoksia, mitkä kohdeorganisaatiossa on olisi tarve tehdä. Nämä muutokset liittyvät ihmisten käyttäytymisiin ja taitoihin, sekä roolien muutoksiin. Tuloksissa on myös kuvattu vastauksia muutamiin käytäntöihin liittyen Scrum:in käyttöönottoon.

Asiasanat: Agile, Lean, Scrum, Waterfall mode, software development

CONTENTS

VOCABULARY

.NET	Software framework developed by Microsoft
Acceptance test	Test to determine that defined requirements or specification are met
ERP	Enterprise Resource Planning
C#	Multi-paradigm programming language developed by Microsoft
IT	Information Technology
Queue Theory	Mathematical analysis of how tasks move through a system with queues
OOPSLA	Object-Oriented Programming, Systems, Languages & Applications
PC	Personal Computer
R&D	Research and Development
Theory of Constraints	Methodology for identifying the most important limiting factor that stands in the way of achieving a goal and then systematically improving that constraint until it is no longer the limiting factor
Unit test	Method for test individual unit from software source code
User Acceptance Testing	Process to verify that solution works for user

1 INTRODUCTION

This chapter presents the background why this work is done. It also describes the specific target for the thesis.

1.1 Background

The target company's R&D high level management set a goal to start planning and moving the R&D organization to use more Agile and Lean approaches on their work, and at the same time start using more standardized processes across all R&D teams. They have chosen Scrum to be the process what every R&D team should use as their development process. The high-level management is expecting this process change to produce more value for end customers by speeding up product development, make it more responsive to new customer requirements, increasing software quality and customer satisfaction.

1.2 Objective of thesis

This thesis focuses on one of the R&D Development Teams and Product Management organization in the target company. The goal was to recognize the most required changes that should be done and new practices when the organization moves towards Agile and Lean practices, and when Scrum is taken into use. These proposals are based on talks with several different people in the organization, studying theories from Agile, Scrum and Lean, and also the using author's experience of software industry.

2 SOFTWARE DEVELOPMENT PROCESSES

This chapter presents the most commonly used software development processes. Firstly, the oldest process called Waterfall model is described. After that, Lean and Agile are explained more deeply, as well as the thinking behind them. Then, the processes based of them are presented. This chapter is also presenting why Agile and Lean practices are seen as better approaches in the software development then the Waterfall model.

2.1 Waterfall model

The first software development process is referred to as the Waterfall model. This model's history goes back to the software development conference in June 1956, where Herbert D. Benington was in his presentation describing methodologies how Semi-Automatic Ground Environment (SAGE) – system was developed. (Ashmore & Runyan 2014, chapter 1; Benington 1983.) In 1970 Winston Royce published the paper "Managing the Development of Large Software Systems" and shared his similar kind software development methodologies as Benington and this is more commonly referred to as the first description of the Waterfall model (Craig & Basili 2003).

This model is originated from other well-defined processes, for example the processes used in construction and manufacturing. In these, each phase of the process is well defined and understood, and each phase needs to be completed before moving to next phase. (Crookshanks 2014, chapter 6.) Figure 1 shows a typical Waterfall model used in the software development.

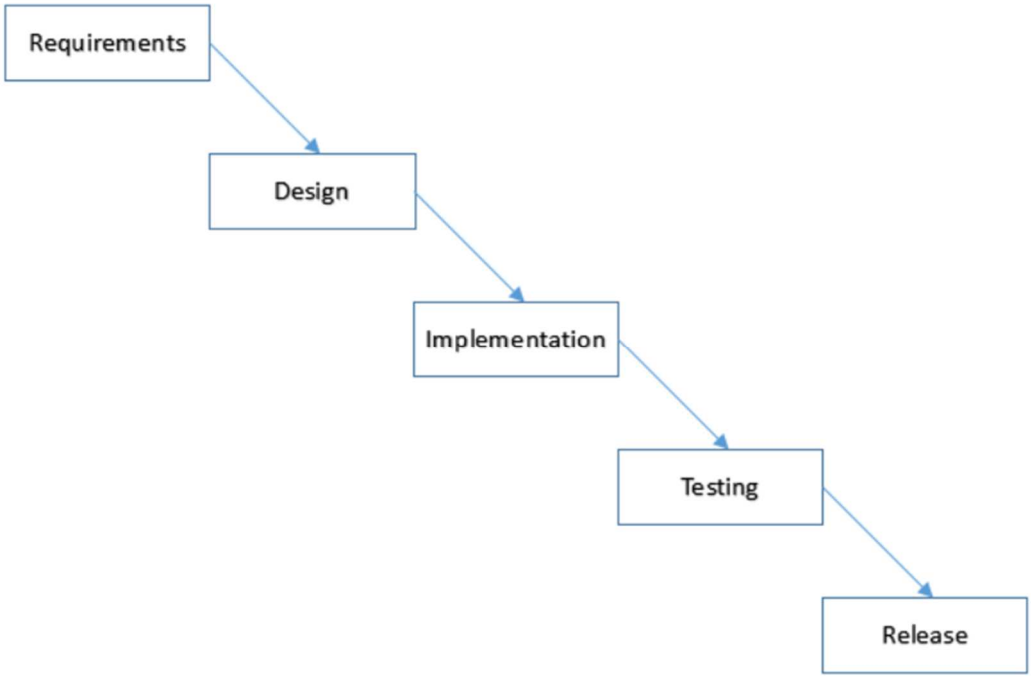

FIGURE 1. Typical waterfall model

In the beginning, all of requirements must be fully documented and acknowledged to completed list of the desired functionalities. Then the process moves to a design phase where a technical team design system based on these requirements. This design is usually reviewed by a management to ensure that all of the requirements are met in the design. After the design phase is ready, coding phase can be started to implement the requirements. When the coding phase is ready, a code can be moved to a testing and a verification phase, which is usually done by a dedicated testing team. In this phase, the system is tested that it meets all of the requirements and the system behaves correctly. When the testing phase is completed, the system can be released for users. If the requirements are changed after the requirement phase, it is usually handled as a change requirement following the same Waterfall model but only with this single item. (Crookshanks 2014, chapter 6.) Even there is more and more demand in the software development to move all of projects from Waterfall model to start using the Agile approaches, in certain types of the projects could still be more beneficial to use Waterfall model approach. Good candidates for these kinds of projects are that there is confident to be able to prepare a highly defined and set requirements, and there is a very low risk that changes will happen. (Davis 2013, 74-75.)

2.2 Lean

Origins of Lean thinking starts in 1945 from Japan. There a small car company, Toyota, with its manager Kiichiro Toyodar had a problem how they could meet results of the American car manufactures. At that time, the Japanese economic situation was not a strong and the people did not have much money to spend so there was a need to manufacture cheap cars. On that that time, a mass production was a known way to achieve this and using it would have mean to create thousands identical parts to gain economies to scale but there was lack of having enough manufacturing materials, amount of need of cars was small and variety was in demand. Toyodar had a vision that the parts should arrive on an assembly line only when there was need "Just-in-Time" and this should not be accomplished by warehousing but manufacturing the parts just before they are needed. (Poppendieck & Poppendieck 2003, Chapter 1; Poppendieck & Poppendieck 2006, Chapter 1.)

Taiichi Ohno, a machine shop manager in Toyota, was a person who responded to Toyodar's vision and challenge by developing to be known as the Toyota Production System, TPS. He based on his ideas for the TPS from studying one of the American car manufactures, Ford Motor Company, and gaining insight from how American supermarkets handles their inventory. To these finds he joined his knowledge of spinning and weaving, and ideas of his workers in machine shop. A fundamental idea in Ohno's thinking was to eliminate a waste. In his mind anything, which did not produce a value for customer was the waste. For example, a warehousing part, if it was not needed immediately, or having any wait in the process or having any extra step in the process is the waste. He believed that it is better to wait an order than build up the inventory and deliver a product immediately after an order. The Lean, as a term, was firstly introduced 1990 in a book named "The Machine That Changed the World", written by James Womack, Daniel Jones, and Daniel Roos. It was the new term what used to be called the Toyota Production System. (Poppendieck & Poppendieck 2003, Chapter 1; Poppendieck & Poppendieck 2006, Chapter 1.)
Principles underlying in the Lean can be mapped to be used in the software development. On following are gone through these mappings.

2.2.1 Eliminate waste

Everything from the Lean point of view, which does not add the value to the product for the customer is the waste. Seven reasons have been recognized in the Lean manufacturing for the wastes and these have been transformed to the wastes in the software development. This transformation can be seen in table 1.

TABLE 1. The Seven Wastes (Poppendieck & Poppendieck 2006, Chapter 4).

The Seven Wastes of Manufacturing	The Seven Wastes of Software Development
In-Process inventory	Partially done work
Over-production	Extra features
Extra processing	Relearning
Transportation	Handoffs
Motion	Task switching
Waiting	Delays
Defects	Defect

A partially done work is an inventory of the software development and it has tendency to come more obsolete as longer it stays an undone work. A big problem with the partially done work is that until it is not integrated with a whole system, no one can tell if there will be possible problems with it and until the work is not delivered to the production, no one can tell will it really solve a business case. The partially done work can also carry financial risks as it ties resources that have not yet yield a result. When ending up in a situation that this undone work is no longer required, there may be sometimes required to do big invests to write it off. Examples for the partially done work are a uncoded documentation, an unsynchronized code, an untested code, an undocumented code and a undeployed code. (Poppendieck & Poppendieck 2006, Chapter 4; Poppendieck & Poppendieck 2003, Chapter 1.)

An extra feature is something which is not needed to get the customer's current job done. If there is not a clear and economic need for the feature, then it should never be developed. Every code in the system is required to be tracked, compiled, integrated and tested every time the code is touched and it is required to maintain for the life of the system. Every new code increases the

system's complexity and will come as a potential failure point. There is also a possibility that this added extra feature will never be used as there was not a really need for it in the first place so putting it in the system is the waste. (Poppendieck & Poppendieck 2006, Chapter 4; Poppendieck & Poppendieck 2003, Chapter 1.)

Relearning, as a word itself describes, is something which is needed to learn again. It is well-known in the software business that everybody should remember what they have learned but in many times, approaches to capture a knowledge is far too verbose and far less rigorous then it should be. Another way to produce the waste here is forgot to utilize the knowledge and an experience what people gained through their working career path by not engage them in the development process. (Poppendieck & Poppendieck 2006, Chapter 4.)

When the people are working on tasks, they are gaining a tacit knowledge on it. This tacit knowledge is something, which is very hard to move with the documentation and every time when a handoff work is done to another person, some of the tacit knowledge are left on the mind of the originator and will never move on the other person. If there is made a very conservative estimation that the each of the handoffs will affect losing the fifty percent of this tacit knowledge, for example after the five handoffs there is left only the three percent of the original tacit knowledge. (Poppendieck & Poppendieck 2006, Chapter 4.)

The software development is a work, which requires a high level of concentrated thinking. Every time when software developers are required to change the tasks, it does not only distract them but also will require them a time to orient on the new task resulting detract from both the tasks. When the time used to complete the tasks are wanted to kept a minimal, it best to work at one task from begin to finish and then start the new task. (Poppendieck & Poppendieck 2006, Chapter 4; Poppendieck & Poppendieck 2003, Chapter 1.)

In the software development work, large cause of affecting delays is to have to wait for people working on other areas to be available. Software developers are required to make a critical decision very often and it is very naive to think that all of the necessary information is found on the documentation. It is possible to make a quick decision if the developer has a good knowledge about what are wanted accomplish with a current work and if there is a person available very close by to who can answer questions. If these requirements are not met, the developer have three options to do: stop working and try to find an answer, switch a task or just guess and continue working. If

finding the answer has a high hassle factor, the developer usually chooses the option two or three. If there is not much of the penalty of waiting the answer, then the developers are likely to spend a good deal of time for waiting the answer. Complete, collocated teams and short iterations with regular a feedback can decrease the delays. Most important thing still is to make sure that the knowledge is available exactly when and where it is needed. It is good make the decision as late as a possible to prevent requiring to change it but not too late as then the decision is ignored. (Poppendieck & Poppendieck 2006, Chapter 4; Poppendieck & Poppendieck 2003, Chapter 1.)

Any of the defects found in the software development can be considered the waste. How big amount of the waste it is affecting is correlating how big the impact of the defect has and how much a time goes by before the defect is detected. A critical defect found very quickly is not affecting as much as the waste as a minor defect, which is founded many weeks later. Preventing getting the defect on code, every code base should have included acceptance and unit tests. Moving doing these tests on the beginning of the development has also a deeper reason than only a mistake-proofing. The acceptance tests are on they best when they constitute the design of the product and match on the design to the structure of the domain. Writing the unit test before coding leads to a simpler, a more understandable and a more testable code, and they also constitute the best documentation of the system because they are always needed update to pass. Still the acceptance and the unit tests only proves us, that the code works as we think it does and does not fail how anticipated. The software has devious ability to find ways to fail, so testing experts with the good skills of the exploratory testing are required to test the code early and often. (Poppendieck & Poppendieck 2003, Chapter 1; Poppendieck & Poppendieck 2006, Chapter 4.)

2.2.2 Build quality in

To achieve building a quality in to the software means putting the quality into the code right from the beginning, not test it in later. A main focus should not be putting the defects in a tracking system as this can be seen as the partially done work, which is the waste, but there should be a process to avoid creating the defects in first place. (Poppendieck & Poppendieck 2006, Chapter 2.)

A good practice to build the quality is to use a test-driven development. In this method, the developer is writing a one automated test, then write enough of the code to make this one test to pass

and then refactor the code to primarily improve readability and removing duplicates. Using a continuous integration approach, these tests and the code are integrated into the whole system as often as possible to run builds and the tests to see that new defects are not introduced. If the builds or the tests are failing, no new code is added until issues are fixed. End of the day, longer and complete testing is run, and in every weekend the system is attached even more complete testing harness. (Kniberg 2007, 82, 84-85; Poppendieck & Poppendieck 2006, Chapter 2.)

2.2.3 Create knowledge

In the Waterfall model development, there is existing a fact that all of the requirements should exists first before coding can be started and they are separated from the code but because the software development process is a knowledge-creating process and before any of the real implementations are made, the requirements are required to be changeable. With an early design, it is nearly impossible to fully anticipate the complexity of the encountered during implementation and the early design does not account a feedback coming from stakeholders or the customers. A better approach is to use a scientific method by establishing hypotheses, conducting many rapid experiments, creating a concise documentation and implementing a best alternative. Future should not be guessed and called it a plan, instead there should be used the feedback based on reality to develop a capacity to rapidly respond to the future as it unfolds. (Measey & Radtac, 154; Poppendieck & Poppendieck 2006, Chapter 2.)

Every Lean software organization is knowing that they should be all of the time improving the processes because in complex environment's like the software business, there will be always problems. There should exists a process for development teams to systematically learn and improve their own processes and giving them responsible to make necessary changes. (Poppendieck & Poppendieck 2006, Chapter 2.)

2.2.4 Defer commitment

People generally wants to get tough decisions out of way to reduce a number of the unknowns. When working with a complexity, what the software development is, with an uncertainty, it is a more successful approach to tackle tough problems by experimenting with various solutions, and learn as much as possible before making irreversible decisions to make it as late as possible. It is easier

to change a decision what have not made yet. Above all, there should be aim to make most of the decisions reversible. When using an iterative development, this kind of approach will lead more an adapting design as the decisions can be made and then easily changed when changes are occurring during the development. The software system does not need to have a complete flexibility but it does need to have options to make the changes so a system architecture should be designed to support the addition of the any product backlog item. (Measey & Radtac, 154; Poppendieck & Poppendieck 2003, Chapter 3; Poppendieck & Poppendieck 2006, Chapter 2.)

2.2.5 Deliver fast

The customers usually like rapid deliveries as it allows possibility to delay their decisions and it is meaning a quicker gratification. For companies, who can deliver faster than their customers change their minds, it means having less resources tied up in a work-in-process and this reduces a risk. When the developers are developing a lot of the code without testing, the defects will pile up. When the code is developed but not integrated, the high-risk part of the effort usually remains and when the system is completed but not delivered in the production, the risk still remains. All of these risks can be reduced significantly by shortening a value stream and this means delivering faster. (Poppendieck & Poppendieck 2003, Chapter 4.) The value stream can be shortened by driving down a cycle time with using small batches, restricting a work in progress and limiting the size of the work (Measey & Radtac, 154).

2.2.6 Respect people

A Toyota Production System's operational value stream to the customers builds on four cornerstones: System Designer Entrepreneurial Leaderships, Responsibility-Based Planning & Control, Set-Based Concurrent Engineering and Expert Engineering Workforce. From these four, three are concerning the people involved in the production. (Poppendieck & Poppendieck 2006, Chapter 1, Chapter 2.)

System Designer Entrepreneurial Leaderships: A company is respecting its people by developing good leaders who ensures that teams have a leadership, which is fostering engaging, thinking people and focusing their efforts on creating great products. (Poppendieck & Poppendieck 2006, Chapter 2.)

Responsibility-Based Planning and Control: Respecting the people means that the teams are given general plans and reasonable goals and are trusted to self-organize to meet the goals. Respect means that instead of telling to the people what to do and how to do it, there is developed a reflexive organization where the people use their heads and figure this out for themselves. (Poppendieck & Poppendieck 2006, Chapter 2.)

Expert Engineering Workforce: The company needs to ensure that an appropriate technical expertise is nurtured and the teams are staffed with expertise to accomplish their goals on areas where the company expects to have a competitive advantage. By buying all the expertise, the company will find out that competitors can buy it also and seeing no need for investing the expertise, they will have no sustainable competitive advantage. (Poppendieck & Poppendieck 2006, Chapter 2.)

2.2.7 Optimize the whole

The Lean organization should focus on optimizing the whole value stream and not the part of it. If only the part of the value stream is optimized, there is a high possibility that the overall value stream will suffer. Highly well sub-optimized parts individually do not necessarily make a high performing whole system as it will come how well each part of the system will work together. (Poppendieck & Poppendieck 2003, Chapter 7; Poppendieck & Poppendieck 2006, Chapter 2.)

2.3 Agile

For decades, software engineering practices rested on the practices such as the Waterfall model, trying perfectly to define what is wanted to build and then build it. On the way, a software industry was introducing more documents, specifications, timetables, schedules, processes, milestones, etc. and results started to be even worse. At the same time, there were few IT projects where performance was much better. There were small teams that could deliver thousand lines of a working code in every week. When these small teams' project practices were traced and studied, results showed that the practices were totally different then what a main software industry was using. These teams were self-organizing to produce a working software on short iterations, obtaining a frequent feedback from a client and then rapidly adapting and extending the working software to

new requirements. The teams produced a high-quality software and they were using several strategies to automate a process such as automating testing, building, and a deployment. The new set of these practices and the tools started to emerge from these teams, including a pair programming, short releases, the automated building and the testing, and a test-driven development. (Medinilla 2012, 35-36.)

There were not only pioneers empirically experimenting the Agile practices but some of the pioneers were to investigating a Lean body of knowledge and many of those bumped into a 1986 published seminal paper by Ikujiro Nonaka and Hirotaka Takeuchi "The New Product Development Game". In this paper, there was said that to survive in the software business, it was not anymore enough to have a good product and a price but the customers were requiring the companies evolve their products for changing a customer needs. Nonaka and Hirotaka were founding that the companies who followed the Waterfall model practices were not able to produce best possible results. Cross-functional teams of the people with different skill sets that were iterating and adapting the requirements, a design, building, and the deployment at the same time were achieving the best results in terms of a creativity, an innovation, a productivity, a quality, and the time to market. (Takeuchi & Nonaka 1986, referee 9.10.2016; Medinilla 2012, 37.)

In the 1990s, there were experts who started to apply Queue Theory and Theory of Constraints to the software development. They were dividing a project in small batches and trying to move them through the development process as fast as they could by removing bottlenecks. At the same time, there were scientists studying complexity science theorized that the software development is a complex field, meaning that the requirements are not are staying stable during the project. They were founding that the predictive method of the requirements gathering and a full solution design is not adequate but inspect and adapt or empirical method should be used to approach the software development as a complex problem, and the products should be built iteratively and incrementally. (Medinilla 2012, 37-38.)

These were the times when new software development practices started to be born. Most important ones were eXtreme Programming (XP), Scrum, Feature-Driven Development (FDD), Crystal Methodologies, or Dynamic Systems Development Method (DSDM). (Medinilla 2012, 38.)

The Agile is just a common dominator and any method or approach that follow its values or a principle can be considered the Agile. The term is not owned by anyone so there is existing many interpretations from it. (Kniberg 2011, 105-106.)

2.3.1 Born of Agile software development and Agile Manifesto

The Agile software development was invented when seventeen thought leaders from a software community met to discuss how to succeed with the software development. Many of these leaders were the experts who have been creating previously mentioned new software development practices. During of these meetings, these seventeen experts found a shared vision of how to succeed with the software development known as an Agile Manifesto. (Kniberg 2011, 104; Medinilla 2012, 38.) This manifesto is seen in figure 2.

FIGURE 2. Agile Manifesto (Agile Manifesto 2001, refereed 16.10.2016)

Result of the meeting were also formed twelve principles behind the Agile Manifesto values:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a reference to the shorter timescale.*
- *Business people and developers must work together daily throughout the project.*
- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- *Continuous attention to technical excellence and good design enhances agility.*
- *Simplicity--the art of maximizing the amount of work not done--is essential.*
- *The best architectures, requirements, and designs emerge from self-organizing teams.*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. (Principles behind the Agile Manifesto 2001, refereed 16.10.2016.)*

Ángel Medinilla has been doing analysis in his book "Agile Management, Leadership in an Agile Environment" about the creators' thinking behind the manifesto. According to him "We are uncovering better ways of developing software by doing it and helping others do it" statement is meaning that the creators are indicating their expertise here from the software development and saying that this not just an academic work. They have been experiencing challenges in the software development, tested their own ideas and helping others to do it. They are sending a message that is not only something, which works only for them but it can work for all others too. (Medinilla 2012, 39.) Medinilla has been doing also an analysis from four foundational values in the Agile Manifesto. From "individuals and interactions over processes and tools" he is saying it is not meaning to ban the processes and the tools but both these should be designed a way that they are supporting personal interactions. The processes and the tools should foster a communication, an interaction and a face-to-face communication. (Medinilla 2012, 39.)

From "Working software over comprehensive documentation" he is saying that the project documentation in the Agile and also in the Lean point of view is a waste. There should be a practice to

have as few documentations as possible because creating more of it does not give more functionality or a value for the customers but the documentation should be still created. It is not best to spend first months in the project to creating the documentation but it should be used for designing prototypes, proof of concepts or technical spikes and evolve through this into full a product in short client-driven iterations. (Medinilla 2012, 39.)

On the third value "customer collaboration over contract negotiation" Medinilla is saying that the contracts are important but if a client is finding new needs and these are not put into the product because in the contract these are not required, the most possible value is not provided to the customer. Better way of collaborating and building the product is reached when both parts accept that the project will change and are fine with this. (Medinilla 2012, 39.)

In the last value "responding to change over following a plan" the acceptance of the change is also present according Medinilla. He is saying that planning is important but there is required to have understanding that planning will be outdated immediately and there is need for constantly adapt to a changing environment. (Medinilla 2012, 39.)

2.3.2 Scrum

The Scrum is the most known and used software development approach used by Agile teams. It was created by Jeff Sutherland and Ken Schwaber, and the Scrum was first time introduced by them at the OOPSLA conference in 1995 (Schwaber & Sutherland 2016, 11). On following is gone through practices and guidelines behind the Scrum.

2.3.2.1 Scrum team

In the Scrum, a team has three main roles: a Product Owner, a Development Team and a Scrum Master. The Product Owner has responsibility to maintain and prioritize the product backlog of the features and requirements what the team is required to build. The Product Owner is required to ensure that the product backlog is clear to all, it is showing what the Scrum team is required to work next and backlog items are understood by the team members for level needed. These tasks can be done by the other Scrum team members but the Product Owner is still always accountable for them. Entire organization is required to respect the Product Owner's decisions and this is meaning

that no else is allowed to ask the Development Team to work for the different requirements and the Development Team is not allowed to act on what anyone else is saying. The Product Owner's role is not a committee role but is a one person's role. (Greene & Stellman 2014, chapter 4; Schwaber & Sutherland 2016, 5.)

The Development Team consist software professionals who are responsible for doing the development work. This team is a self-organizing, meaning no-one cannot tell them how to create increments of potentially releasable functionality from the product backlog. The team has cross-functional skills to create the increments and the Scrum does not recognize any other titles for the team members than the Developer. There should not be sub teams in the Development Team. Individual persons may have special skills but accountability for work belongs to the whole team. (Schwaber & Sutherland 2016, 6.)

The Scrum Master's role is to keep the Scrum team's work ongoing and making sure that the Scrum team adheres to a Scrum theory, practices, and rules. This role helps the Development Team for removing impediments preventing the team's progress, coaching them to be self-organizing and helping them to create a high product value. Towards the Product Owner, the Scrum Master helps managing effectively the product backlog, helping to understand need for a clear and concise Product Backlog item, and facilitating Scrum Events. Towards to the organization, the Scrum Master helps adopting the Scrum practices, causing changes to increase the Scrum Team's productivity and working with the other Scrum Masters to increasing effectiveness of the application of the Scrum. (Schwaber & Sutherland 2016, 6-7.)

2.3.2.2 Scrum events

Key of the Scrum is a Sprint. It is the time box of the one month or less where potentially released product increments are created. Every Sprint starts with a Sprint Planning session where is selected what new product features will be created during the Sprint and how this work can be achieved. The result of the Sprint Planning comes a Sprint Backlog, where all of this info is. (Greene & Stellman 2014, chapter 4; Schwaber & Sutherland 2016, 8.)
During the Sprint, in each of the work day is kept a 15-minutes time-boxed event called a Daily Scrum. This event is used for synchronizing activities happening in the team and creating a plan for next 24-hours. Every Development Team members answer three questions in these meetings:

- What I have done since last meeting?
- What will I do until the next meeting?
- Do I see any impediment that prevents me or the Development Team? (Greene & Stellman 2014, chapter 4; Schwaber & Sutherland 2016, 11.)

The Daily Scrum helps inspecting the progress of the Sprint and optimizing probability to for the Development team to meet the target set on the Sprint Planning. The Scrum Master is enforcing a rule that the Daily Scrum is kept only 15-minutes long and only the Development team members are participating on it. After meeting, often the whole Development team or the team members meet for detailed discussions, or to adapt, or re-plan, the rest of the Sprint's work. (Schwaber & Sutherland 2016, 11.)

End of the Sprint is kept two events, a Sprint Review and a Retrospective. The Sprint Review is a meeting where a working software is demonstrated for the Product Owner and stakeholders. In this meeting, there is also discussed by the Development Team what went well and what problems they were encountering during the Sprint. Entire group collaborates with a topic what would be most valued to do next and by this way there is provided a valuable information for next the Sprint Planning. (Greene & Stellman 2014, chapter 4; Schwaber & Sutherland 2016, 11-12.)
The Retrospective is a meeting, where the whole Scrum team is participating for figuring out lessons what they learnt from the Sprint. They inspect what went well during the Sprint and what would be potential improvements. The result of the Retrospective should produce a plan what identified improvements they could implementing during next the Sprint. After these two end events, the Sprint is ended and a new Sprint can be started. (Greene & Stellman 2014, chapter 4; Schwaber & Sutherland 2016, 12-13.)

2.3.2.3 Definition of Done

Every Scrum Team member needs to have same understanding what it means when the work is completed to ensure transparency. Conceptually, the definition of done is a checklist describing a work what a team is expected to be completed before the team can declare its work to be potentially shippable. When there are multiple Scrum Teams working with a same system or release, they must mutually define the definition of done. (Kenneth 2012, 74; Schwaber & Sutherland 2016, 16.). Figure 3 shows an example of the Definition of Done checklist.

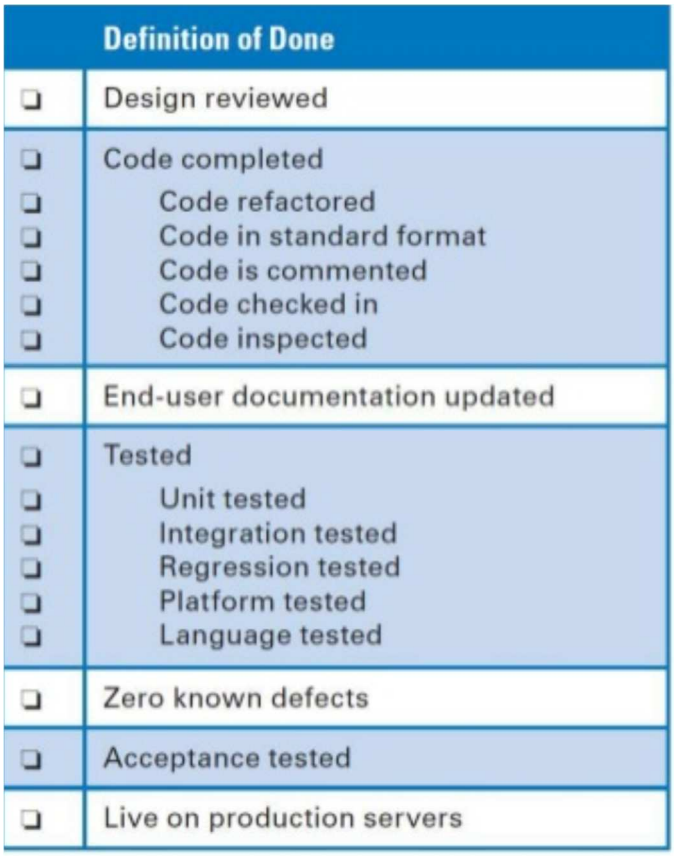

FIGURE 3. Example of the definition of done checklist (Kenneth 2012, 74.)

2.3.3 Kanban

A Kanban is an Agile software development practice using the Lean approach. A word "Kanban" is Japanese and means a "visual card". In Toyota, a term used for a visual and a physical signaling system that ties together a whole Lean production system. In 2004 David Anderson was pioneering the Lean thinking and the Theory of Constraints to the software development process. Under the guidance of the other experts there were found "Kanban system for software development", which is mostly only referred as the Kanban. (Kniberg 2011, 112.)

The Kanban is based on three principles and first one is visualizing a workflow. In this, task is to split all of the work on pieces and add on a card, for example sticky notes. Then there is created a board by adding a column for the each of the phase where the work cards can be and then the cards are put on the correct places of the board. In this way, all of the work information is visible for everyone at whole time and everybody will see what a real work process looks like. (Hammaberg & Sundén 2014, 49-50; Kniberg 2011, 113.) Example of the Kanban board can be seen in figure 4

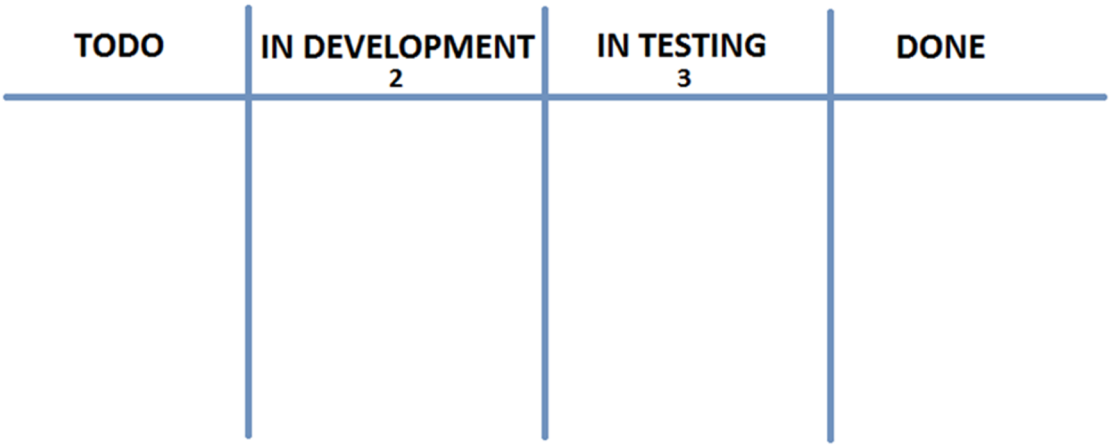

FIGURE 4. Kanban board

The second principle is Limit Work in Process (WIP). In this, task is to add numbers on the columns indicating how many of the work items can be in it at the same time. Aim here is to make a cycle time lower, the work items flow faster through the board when less of the items are worked on the same time. Generally, a lower number is better as it should be lead to the lower cycle times but setting it correctly are depending on how much there is a pressure for continuously improve, what is number of the people on the team and their availability, and what kind of items are the working items. Making a limit too high will lead that the work will be idling and making it too low will lead that people will be idling. (Hammaberg & Sundén 2014, 50, 110-111; Kniberg 2011, 113.)

When the Kanban process is taken into use and the flow is visible, it will be always indicating bottle necks. Here comes a third principle necessary to take in use, manage the flow. This is a task, where a goal is to make the work items flow quickly and without interruptions through the process. Helping to manage the flow and removing the bottle necks can used inspirations from the Lean thinking to reduce the waste by reducing waiting times, removing blockers and avoiding reworking. Also practices from the Scrum such as the cross-functional teams can help making the flow better. (Hammaberg & Sundén 2014, 50-51, 134-143.)

3 TARGET ORGANIZATION

A target company is a software company developing for a specific industry area integrated information technology and digital marketing solutions. It is one of the biggest on its area and has been developing the solutions for customer over forty years. Currently it has over 27 000 customers and it serves clients in over 100 countries. The company is shared on two major business units; North America and International unit.

A target organization is part of the International business unit. The organization's history starts from year 1983 where it belonged on a very small company. During years through several acquisitions and changes it has ended up being part of this bigger company.

3.1 Target organization product

A target organization's software product is ERP for retailers and importers, and it is part of the Company's ERP offering. A product history starts from middle of the eighties where it was one of the first PC based products on its own industry area and it has been also one of the first Windows based products on middle of the nineties. A product backlog includes issues from customer paid requests, the company's own funded new features and maintenance issues.

3.2 Product management department

A Product Management has main responsible from all of the new developments in the product. The Product Management team has four Product Owners, each having own responsible area of the product and they are managed by a Product/R&D Manager. They are working with a R&D department by defining a backlog what the R&D is required to work on. They are responsible for creating requirements specifications for issues and prioritize issues work order. They also are responsible to work with the R&D to get work estimates for both the new features and the customer requests, and they also analyze a feedback from customer pilot projects. One of the Product Owners is also occasionally working on with development tasks.

With a Sales Department, they work for helping creating marketing materials and making pre-studies/preliminary work estimates. For a Support Department, they are helping to analyze a product feedback coming through support's channels and keeping trainings about the new features functionalities. For a Project Department, they help creating training materials and also train them about

the new features. They also talk directly with the customers for getting a feedback and creating the possible new customer requests. On figure 5 is seen the Product Management's organization structure.

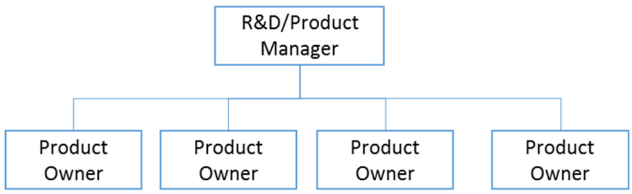

FIGURE 5. Target organization Product Management

3.1 R&D department

The R&D department is divided in three developments teams, a RMD team and a Fast Response Unit. The Development Teams consists Developers, Quality Analysts and Team Leaders, who are leading the teams work and also working with developer's tasks. For each of the Development Team, the Product Management has nominated own Product Owner and the team mainly works in area of the product, which is in nominated for a Product Owner's responsible area but occasionally they are required to work with other Product Owners issues also. One of the Product Owners do not have the own nominated specific team and this Product Owner work is shared between the teams. In some cases, there is also created a new team on a fixed time period by taking persons from the Development Teams so that these persons then work issues on shared responsibility between they own team and on the fixed time period team. The Development Teams work is managed by a R&D Service Manager who is also the supervisor for all of the Team Leaders and the Developers.

The RMD team is managed by a RMD Manager. This team's main responsibility is to run the pilots for main new features with selected customer. The team is also responsible for testing and a test automation as all of the R&D Quality Analysts are reporting to the RMD Manager. A specific automation development work is created outside of the Development Teams and located in the company's other office in an own specific team. Help documentations and localizations are produced in this team as these area experts are reporting to the RMD Manager.

The R&D department's small unit called the Fast Response Unit is used to help the Support Department to deliver a promised response time for the customer in severity maintenance issues. The

Fast Response Unit consist two developers and they do not have the Team Leader like the Development Teams have. Their backlog is mostly created by The Support Department and managed by a Support Manager, testing is done for their issues in Support Department. The Fast Response Developers are reporting for the R&D Service Manager like the Development Teams Developers.

The whole R&D department is managed by a R&D Manager. This role's main responsibility is to supervise all of the R&D processes, create a customization budget for the R&D developments, help and make prioritization with the Team Leaders and the Product Owners to create the development backlogs for the teams, and keep the R&D aligned with the company's R&D strategy. The R&D Manager reports to the Product/R&D manager, same as all of the Product Owners. The R&D organization structure is seen on figure 6.

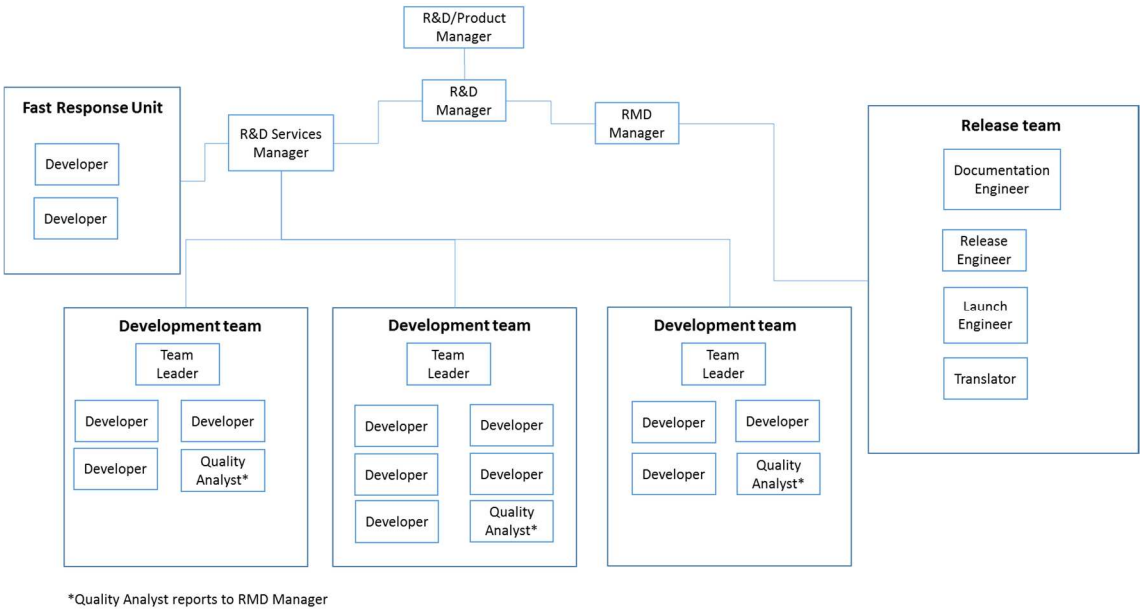

FIGURE 6. Target organization R&D structure

4 CURRENT R&D WORK PROCESS

Current R&D and Product Management processes are mostly based on the Waterfall model. Before moving to the next phase, the previous phase is tried to define precisely and move only to the next phase when previous is ready. On following are described more closely how the new feature and maintenance processes works, and what kind of a release process is used.

4.1 Own funded new feature/Change request process

In each of the teams, the Team Leaders and the Product Owner are got communicated from the Product/R&D Manager or the R&D Manager for getting the new feature for the team to be developed. The Team Leader then assigns tasks for creating specifications documentation and time estimation for one of the team's Developer. In many cases, the Team Leader is a person who is making the specifications and time estimations tasks as they are recognized also having a technical leadership in the team. In some cases, the Product Owner may have already defined the person responsible for the case and then work will to this Developer. After the Developer has created the specifications documentation and time estimation, these are given back for acceptance to the Product Owner. If the new feature is a change request, it will be send for customer acceptance after the Product Owner's acceptance, otherwise Product Owner will be an only acceptor. After the acceptation phase, an implementation task is usually tried to give for the Developer who originally created the specifications and the time estimation but if the Developer is not available, it can be given for some other developer even in other Development Team. If the specifications and time estimation creator was the Team Leader, then it is the Team Leader's decision to decide who will work on implementation when the Team Leader' is not available for the work.

After the implementation phase is done, the new feature is moved for the team's Quality Analyst for testing. The new feature is tested against the created specifications documentation. If the new feature's or the customer change request's a work size exceed a certain level, then final testing and acceptance are done in a User Acceptance Testing phase. In this phase, the new features are tested with the Product Owner and the change requests with the customer. In the end, the new feature or the change request is released on a next monthly release.

4.2 Maintenance process

The Support Department receives maintenance issues from the customers. They are making analyzes for the issues and if the issue is recognized as a bug, they will give for the issue give a priority and a severity, and moved to it the Fast Response Unit's backlog. Most of the maintenance issues are handled by the Fast Response Unit team but depending of the bug's priority and the team's current capacity or if the maintenance issue is strongly related on a feature what R&D Development Teams have recently developed, the issue can be moved to the R&D Development Teams to be fixed. The R&D teams handle the bug fixing same as with the new features and the change requests work that Team Leader assigning the work to the Developer.

If the Support Department recognizes that the issue is not a maintenance issue but a new feature, they are communicating with the customer and if the customer wants this new feature developed, it is moved for the Product Management to be handled as the new feature.

4.3 Release process

The release process consists three phases: a development, a feature freeze and a release phase. On the development phase, new coding and testing are done. When the development phase is completed, the new feature is moved on the feature freeze phase and on this phase, is only allowed make bug fixes for the feature but not anymore, any new changes. When the feature freeze phase is completed, the new feature is release.

On the time point view, the organization follows a monthly based release cycle and in the middle of every month, a new release is made available. The earlier described phases also are mapped on the organization's coding branching strategy where the development phase is done on a trunk branch, the feature freeze phase on a feature freeze branch and release phase on a delivery branch. The trunk branch is existing a whole time, the feature freeze branch is created on the beginning of every month and it will replace a previous delivery branch when the release happens. The development phase for certain new features may last over the several release cycles and preventing under development features appearing in the product there is used a control data feature, which can be used to enable the new feature visible only when it is fully developed.

The maintenance issues releases depend on the issues severities. If the issue is a very critical issue, it will be developed on the trunk branch, moved from there directly also to existing the delivery branch to be available immediately so that it can be delivered fast as possible. When the issue's severity is not critical, it will follow the standard release process by releasing it on the next coming monthly release after the bug fixing is completed.

5 IMPLEMENTING AGILE AND SCRUM

To move the organization towards using the Agile and the Lean approaches, and use the Scrum practices there is need for make changes and implement new practices as many of currently existing ones does not fit any more on the Agile and the Lean approaches. To recognize these changes and the new practices there were used approach to talk with several people in the organization to get understanding what their level of the knowledge is in the Agile/Scrum and where they see problems, and same time researching theories written by Agile/Scrum experts. Here were also utilized the author's own experience learnt through working on a software business for analyzing the currently existing behaviors and the practices.

On following are described topics what raised through this work. There is not gone through fully the Scrum guideline's topics or described huge number of topics. The topics covered here are the ones, which are seen the most required ones to get a transformation to be started and this is also supporting the Agile's mindset to use iterative approach and not to specify all requirements.

5.1 Self-organizing teams

The Self-organization is one of the key in things to achieve for the Agile based organization. For achieving the self-organizing teams there are three conditions required to be met: an empowerment, a continuous improvement and a cross-functional team (Howard & Rogers, 2011, 27-28). Currently the processes and the behaviors are supporting the way that management roles are commanding and controlling roles, and the team members wait for work. The Product/R&D Manager and the R&D Manager are telling to the teams what new tasks or features they are required to complete and many cases they are making this assignment with info who of the team members should complete this task. There is not existing a process how the teams are constantly thinking how to improve themselves and inside the teams there is lack of the competences to have required skills to be able to complete the tasks coming for them.

5.1.1 Empowerment

When the management wants to support the empowerment teams, it requires them to learn to be Agile managers. The Product/R&D Manager and the R&D Manager role needs change on more only to specifying goals, setting boundaries and to be responsive to the teams when they require a support. Otherwise, the teams are given freedom to operate and make a decision within the set boundaries. (Howard & Rogers 2011, 27.) This change may rise two main challenges for the managers: lacking of trust that the teams perform adequately and fearing that the teams do not do what the managers are considering best to do. There maybe also in background thinking that the manager role and a status will disappear when she or he stops commanding people, and what she or he will do with all of the time what the manager used to spend on commanding the people. (Medinilla 2012, 100.)

The managers are required to understand that when they are delegating their power to the people they are not decreasing any of their power. The managers are still accountable for tasks they are delegating as they are making decisions to delegate and will be on the end responsible for results. Using the delegation is actually creating more the power and the status for the managers. When she or he is managing the people, who has the power to make the decisions, they are managing more an empowered and powerful people and this will lead more increasing the power and the status for the managers. (Medinilla 2012, 100-101.)

To be able to perform as the Agile Manager, there is required to able to provide coaching for the Development Teams. Problems what the software Development Teams faces often do not stand up to a machine model what many of them have used in the past. In the machine model, there is believe that there can be taken an any complex problem, break it down into its component parts, create each of the component more or less separately, and then bring all of them back together in a one final integration effort. Instead, the problems they are facing, are slippery and have never been solved to the people's satisfaction before. At the same time, a world is more uncertain than anyone can remember, and the latest changes will rock the team before they have even settled down from a last change. Also, the people want to know that they matter. They want to know that what they put their effort and thought into yields something valuable and that their contributions to it are seen as valuable too. Coaching will help in both of these areas, producing products that matter in the real, complex, and uncertain world, and adding meaning to the people's work lives. The Agile is far more than an alternate project management methodology. It is great for that, but

that is also the weakest expression of it. Only when Agile is done well, it focuses the teams to create critical products and makes it possible for them to create those one after the another in a way that allows the teams to meet their own high standard of the excellence and pursue a vibrant personal pursue. This means that the teams need coaches who bring to them the clear view of the Agile done well and a host of other skills to make the Agile come alive for them. (Adkins 2010, 4-5.) Although there is required to provide coaching for the teams, the coach does not necessary have to be the Product/R&D Manager or the R&D Manager. Thinking for the managers being only ones' good for coaching comes from a traditional hierarchical thinking that the managers have higher competency than their subordinates but from the complex systems perspective this is not a true. Responsibility can be delegate and empowered for senior people to coach junior people. The Product/R&D Manager or the R&D Manager does not even have to coach for the senior people but there could be considered to hire an external consultant for this task. The only task for the Product/R&D Manager or the R&D Manager would be to figure out which persons, internal or external, would good the coaches. (Appelo 2010, 232-233.)

5.1.2 Continuous improvement

On the Scrum, a continuous improvement is built in on two inspect- and-adapt activities: the Sprint Review and the Sprint Retrospective. On the Sprint Review all of the Scrum team members, not just the Scrum Master and the Product Owner, should be present on the Sprint Review. In this way. the whole team can describe what they have been accomplished, answer any questions and will get benefits for getting a firsthand feedback. To get best a possible feedback, there should be tough who to invite. When inviting inside stakeholders, such as business area owners and executive management will provide an essential feedback for the team that they are progressing towards an economically sensible outcome. Other organization should be also asked to join as for example sales and marketing people will be excellent source of getting a feedback whether the product is converging on a marketplace success. Inviting the Support organization members will also produce the value as they would get know what of the features are developed and they would also give a valuable feedback for the team. Would be also good idea periodically include external stakeholders, such as the customer to the Sprint Review. In this way, the team could get a direct feedback from the customer instead getting it indirectly via the internal stakeholders. (Kenneth 2012, 364-365.)

The second inspect- and-adapt activity is the Sprint Retrospective. This is a most crucial contributor for the continuous improvement from what the Scrum offers. The Sprint Retrospective gives an opportunity for the team to stop for a moment and think after every Sprint. The team is free to examining what is happening, analyzing a way they are working, identifying ways to improve, and making plans to implement these to improvements. Anything that affects how the team creates the product is open to an exploration and discussion, including processes, practices, communication, environment, artifacts, tools, and so on. It is a key that the full Scrum team participates because all of the members have a rich and the diverse set of the perspectives that are essential for identifying the process improvements from the multiple points of the view. Other individual such as the stake-holders or the managers who are not on the Scrum team, should attend the Sprint Retrospective only if invited by the Scrum team. Although transparency is a core Scrum value, the reality could be that the organization have not yet achieved the level of the safety to support non-Scrum team members regularly attending the Sprint Retrospectives. It is mandatory for the team members to feel safety to be able to be have open discussion without feeling inhibited by the outsiders. Without safety, the team members may not reveal real issues and the Sprint Retrospective will lose its effectiveness. (Kenneth 2012, 375-377.)

As there is not currently existing any kinds of official improvements processes in R&D, to make success to the continuous improvement it is essential that both the Sprint Review and the Sprint Retrospective are planned to be started to be use immediately when the Scrum is taken into use. Is also essential that not only people in R&D but the whole organization are needed to be taught sharing values behind these and supporting them because the continuous improvements pro-cesses also do no exists currently on other departments and this way a knowledge about im-portance of these kinds of the processes will be shared.

5.1.3 Cross-functional team

A cross-functional team is a team, which have all needed skills to get an assigned work for them done and produce good quality features. The cross-functional teams are flexible teams by having the team members composed on T-Shaped skills. This kind of members are having a deeper knowledge on certain areas but they also have the skills outside their core competences. (Kenneth 2012, 200-203.)

A current R&D technology portfolio is shared on three main technologies: an old window based development platform, a more modern Microsoft .NET/C# technology and web-based technologies. History of developing the windows based development platform starts few decades ago and many of the R&D members have been in the organization long time, so mostly all of them have skills to develop with this technology.

The .NET/C# technology was added in the technology portfolio some years ago. Implementation was done on way that some of the developers were offered training related this and then they have been the ones who have been developing tasks with this technology. During years, there has been happening natural changes in the organization when people have been moving to other companies and new hires are made, and this have been leading slightly increase in the .NET/C# technology competence. In recently, the organization started on moving on to start developing with the modern web based technologies and currently only few of the R&D people have proper skills to develop with this technology. Currently two of three R&D teams have members with the .NET/C# and web technology skills.

Demand for developing features requiring skills from the NET/C# technology is currently growing rapidly and expectation is that in the near future needs for the web development skills will grow also. To answer this challenge, teams are required to be more cross-functional and to have these cross-functional teams, especially the developer's skillset requires improvements. For all of the developers who currently do not have proper skills with the NET/C# technology, should be offered training for it. After training, every new .NET/C# developer should have a named mentor for helping to start developing with this technology because training can offer only starting point for path with the new technology but it will not make anyone yet having a full capability and with having the experienced tutor during this learning phase will lead it to be more efficient and shorter. The mentor practice could be made even by making changes on the teams' structure so that every team has at least one developer having the skills on the .NET/C# technology. This same practice could be used to answer the challenge with the web technology but as only a few of the R&D members have development capabilities on this, it most likely will require offering even more trainings but first the .NET/C# technology cap is need to solved as there is demand on this technology based implementations rising much more heavily.

To be a high-performance cross-functional Scrum team and keep up with competence challenges, the teams need to find ways to talk and share practices with other people on the other teams

directly. In large Scrum projects and departments there is often formed the communities of the practices to share a knowledge. In these, individuals can meet regularly and not only share common problems but share also solutions what they have been discovering. (Cohn 2009, 210.) This kind of communities should be considered to be formed inside the target R&D department and also considering to form the communities cross all of the R&D departments in the company.

5.2 Working as an Agile team

On the Scrum, there does not exist anymore own work for each of the team member but all of the work is considered as a team's work. The whole team is responsible for having a well-formed backlog, that all of the Sprint backlog items are completed and produced with a quality. (Cohn 2009, 201-202, 204.) Current practices are giving more indication that the R&D team is more of a group of the people then the team. These are mainly seen on practices that the developers usually do not work together but with own individual tasks, the Quality Analyst do not participate an estimation or a planning work and there is a mindset that responsibility of the quality is only within the Quality Analysts roles. These practices are required to change.

5.2.1 Developers working together

One practice to make the developers learn to work more together is having the entire team commit to work only for a one product backlog item until it is finished. This practice would utilize idea from the Kanban that there should be a limitation how much of the work is ongoing on a current phase. Most of the Scrum teams eventually are learning that it is not a good idea to have too many items ongoing, and evolve a culture of trying to get the current items done before starting new items. Eventually limiting ongoing work for the only one backlog item could be very restrictive and using a higher number limitation could be even considered to use in the future. (Cohn 2009, 204; Kniberg & Skarin 2010, 15-16.) For example, limiting a coding work to be always one less than developers number in the team, would make at least two developers always required to work together.
To have more collaboration between the developers could be gained through teaching and implementing a pair programming to the developers. The pair programming is a practice where two developers are working side by side at a one computer, continually collaborating on a same design, an algorithm, a code, or a test. Another one of the developer is a driver and another is a navigator. The driver is the one who is typing at the computer or writing down the design and the navigator's

task is to observe the work of the driver by looking for tactical defects and strategic defects. The tactical defects are such as syntax errors, typos or calling a wrong method. The strategic defects are the ones when the driver is heading down to a wrong path by creating an implementation that would not accomplish what needs to be accomplished. On the pair programming, the driver and the navigator are communicating verbally constantly between each other and can start a brainstorm session easily on-demand. As the developers are collaborating together, it is building trust between them and improving a teamwork. There is also other beneficial gains using the pair programming: the pair of the developers will be producing a code with less defects, they are saving time because the pair of the developers are producing a higher-quality code in about half of the time as an individual developer, the pair developers are happier and happier developers are less likely leaving the company, the developers have a more knowledge about an overall systems especially if they do not pair with a same developer always and learning is enhanced when they see how other developers are approaching a task. (Williams & Kessler 2002, Chapter 1.)

5.2.2 Quality Analyst role

The Quality analyst in the Agile and in the Scrum, are required to be taken within feature planning and estimation process as it is not only Developers' works but the whole team work. When they are left out of these both processes, the team is not making best use of the skills and the knowledge existing in the team and it is not producing then a best possible software. When the Quality Analysts' skills have not taken use into these processes, there is risk that all effects what a new feature may affect to rest of the system are recognize very late on development. (Crispin & Gregory 2009, 32.)

Taken the Agile practices in use there is required to teach the whole team that they are responsible for the quality and not only the Quality Analysts what current practices are supporting. The Agile development has a focus producing a high-quality software in a time frame that maximizes its value to a business and this task is not just the Quality Analysts, the whole team has responsible from it. The Agile team must possess skills to product a quality code to deliver required features and while it may mean having the Quality Analyst being a team member, it does not limit particular tasks to the particular team members and any task might be completed by any of the team members, or a pair of the team members. This means that the whole team takes responsibility for the all kinds of

the testing tasks such as automating tests and a manual exploratory testing, and the whole team thinks constantly about designing a code for testability. (Crispin & Gregory 2009, 15.)

The Quality Analysts are required, when taking the Scrum in to use, to teach to start approaching their work differently how they used to be. Previously, The Quality Analysts have been expecting to be delivered a perfect requirements document and then use it to confirm that systems work as required. Because the Scrum teams are shifting focus in requirements gathering from writing about requirements to talking about them, are the Quality Analysts also required to change their behavior how to find a way how a new feature should behave. The Quality Analysts' key source for finding an answer will become conversation between them and the Product Owner. They should not only limit themselves for the Product Owner and could acquire information when appropriate by talking with the users, customers and other stakeholders. (Cohn 2009, 148 -149.)

The Quality Analyst are also, when taking the Scrum in to use, required to be to learn how to work iteratively. They cannot anymore wait the developer to finish a programming task to be tested as this would produce lengthy delays for the Quality Analysts to get work at the start of the Sprint. When the Scrum team chooses to work with a new feature, they start it with collaborative discussion and the Quality Analysts are required to be part of it. After the discussion when the Developer has enough of the knowledge to be able to start coding, she or he start it and the Quality Analysts can continue discussion with the Product Owner to create list of high-level tests. After this discussion, the Quality Analysts can start creating more concrete tests and a test data. While the Quality Analysts are thinking about the concrete tests, they need to collaborate with the Developers and inform any of the test cases that the Developer may not be considering while he or she is programming. (Cohn 2009, 149, 206-207.)

5.3 Define Definition of Done

Currently Development Teams members have different understanding what it means to say "Done". When going towards to the Scrum practices there is required that the Scrum teams defines a Definition of Done to share same understanding. There are also other three major topics why this should be done. First one is that it will help building a bond between the team members. When the team defines the Definition of Done, it will give to the team members feeling they are all in this together as the team and it is more important to deliver commitments together than focus individual

tasks. Second, it provides a clear communication for stakeholders. When the teams are saying that they will release to a production, the stakeholders have clear understanding through the Definition of Done what it really means and this will drive down the risk of the technical or the other debt being deferred to later in a cycle. Third, it will keep the team on a track and focused. A guesswork will be removed when the team are planning iteration and enabled to them to focus on a delivery instead the speculations of the possible outcomes. (Lacey 2015, Chapter 7.)

When the Definition of Done will be defined, it is important that the whole Development Team's members are involved. This way everybody's knowledge will be utilized and will be giving them valuable building exercise how to work together. Even the Product Owners should be involved as through them, there could be gained companywide or project mandates that the Development Team does not have a knowledge. (Lacey 2015, Chapter 7.)

5.4 Who should be the Scrum Master

On the Scrum, the Team Leader's role comes obsolete and a new role start to exists, the Scrum Master. Most obvious choice would be to make the Team Leaders as the Scrum Master but as these persons might become great Scrum Masters but on other hand they might not be best choices for the role. Currently, the Team Leaders are presenting a technical expertise on the teams but the Scrum Master is not the role in the Scrum Team where the technical expertise is fully exploited to its full potential. A person becoming the Scrum Master should be more considered is an individual willing to take on the responsibilities of the role is about and does the individual have the characteristics required to in the Scrum Master role. (Kenneth 2012, 191.)

5.4.1 Required Scrum Master's characteristics and skills

When looking for a person from the organization, there are certain characteristics and skills what there should be looked for. On a knowledge side, to be an effective process coach for the team, the person must have very a good knowledge about the Scrum. The chosen person does not need to have a technical lead level understanding of the technical issues that the team needs to address and technologies that the team will use to create solutions but a reasonable technical knowledge is an asset. Having a working knowledge about a business domain is helpful but also here the expert knowledge is not required. (Kenneth 2012, 188.)

To be an effectively Scrum Master, the person is required to be able to ask great questions by using their coaching skills in conjunction with their process, technical, and business knowledge. The Scrum Master engage in an intentional inquiry, asking the kinds of the questions that makes people stop and thinking other way and even better options what they might not even though before. A very good Scrum Master almost never directly answer a question but instead reflexively answer with their own question not making it an annoying question, or ask the question for the sake of asking the question, but rather thoughtful, deep, probing question. Doing this, the Scrum Master is helping the people realize that they have the insight to find their own answers. (Kenneth 2012, 188-189.)

As the Scrum Master is not preferring to give out all of the answers, is the chosen person required to be a patient and give the team time to arrive at appropriate answers on their own. The Scrum Master may sometimes to be very tempted give an answer as she or he is knowing the right answer, but it might be that individual just thinks knowing it. The chosen Scrum Master must understand that the collective intelligence of the team is most likely smarter than a one individual and let the team work out the solution, periodically asking probing questions to help guide things along. (Kenneth 2012, 189.)

The chosen Scrum Master is required to have skills how to protect the team. The team will encounter on the way organizational impediments and people with different agendas. The Scrum Master is needed to be well skilled to ensure the protection of the team within the greater context of making economically sound business decisions. When there comes acute sensitivity toward both the team protection and business needs, the Scrum Master helps the Scrum team achieve a healthy balance. The Scrum Master also needs to help the team members who begin to wander away from the Scrum. When things get difficult, it is easy for the people to fall back on a familiar, non-Agile approaches and then the Scrum Master's helps straying the team members overcoming their difficulties by reinforcing how to use the Scrum more effectively. (Kenneth 2012, 189.)

The chosen person must have skills to be transparent in the all forms of the communication. The team members must know that what the Scrum Master is doing and telling is what they will get from her or him; there is no room for hidden agendas. The people expect nothing less of a servant leader. The Scrum Master is also promoting an open and a transparent communication outside of the Scrum team. Without this kind of information sharing it would be difficult for the organization to

inspect and adapt to achieve its desired business results from using the Scrum. (Kenneth 2012, 189-190.)

5.5 Four Product Owners vs three teams

Currently there are more Product Owners than teams, four Product Owners and three Development Teams. This one Product Owner's work is either completed on one of the Development Teams or there is created the fixed new team to complete this work by taking the persons from the current Development Teams and making these persons then work on two teams. Also, sometimes the Product Owner, who has the own specific teams, is working on with the different teams to complete tasks. When looking on the Scrum guidelines, these kinds of the practices cannot continue anymore. The Development Teams should not be working on with any other requirements then their official Product Owner's and there should not be any sub-teams inside the Development Teams (Schwaber & Sutherland 2016, 5).

Firstly, there should be analyzed current Product Owners' responsibility areas and divide them more correctly. There can be seen clear indications that responsibility areas are not divided correctly because Product Owner tasks are sometimes required to work in different teams than Product Owners' official teams. Every Product Owner should also have own team so two of the following practices are required to think: the current R&D organization will be split on four R&D teams or one of the Product Owners is required to change a role. The second option could be made such way that as one of the Product Owner is currently also doing sometimes the development tasks, this person's role is changed. From the Scrum guideline's point view, this person could still at some level do the Product Owners tasks when required as a team's Product Owner's task can be delegated to the other team members but the Product Owner will be in the end accountable for those.

The second options would be here the preferred choice. Using the first options, on the size point of view it would still go on the Scrum recommendation by having minimum three members on the Development Teams and maximum nine members (Schwaber & Sutherland 2016, 6) as there would come three teams having three people and one team having four people. But for a competence point of view, the first options would affect challenge on the teams' cross-functionality as one of the teams would not have the Quality Analysts person skills.

5.6 Determining Sprint length

According Mitch Lacey there are three elements to consider when determining how long the Sprint should be. These elements are a project's duration, customers/stakeholders and the Scrum team. (Lacey 2015, Chapter 6.)

Currently, the target organization's backlog consists own and the customers funded change requests. On the size point of view most of the change requests can be completed during a one-month cycle and rest of the change request are larger but very rarely exceeds over three-month period. Lacey is saying that when the project is lasting less than three months, a four-week Sprint is too long. Using the four-week Sprint would offer for change request stakeholders only possibility to participate two reviews and this would increase risk getting enough feedback during the development. (Lacey 2015, Chapter 6.)

Many of the target organization customers are not familiar working with the Scrum and a Sprint model. They have used the Waterfall-based approach by delivering the requirements, accepting those specifications and then waiting a solution to be delivered them on an end. According to Lacey, in this kind of the environment, the Sprint length have to be longer than a one week as it may be solving some Development Team's problems but it would generate too much stress on customer relationships. Before going on very short Sprint lengths there is required to make education for the customers about the benefits of the short feedback loops. (Lacey 2015, Chapter 6.)

According to Lacey there are three factors relating to the Scrum teams for determining the Sprint length: how experienced the team is in the Scrum, team engineering practices and team's ability for decomposing a work. The target organization's Development Teams have not yet much of the experience about the Scrum and Lacey is suggesting for inexperience teams a rule to use a two or the four-week Sprint lengths. The team's engineering capabilities point of view there is some experience about a continuous integration usage as many of the developers are committing the code on daily but other practices such as the pair-programming and a test-drive development are not generally used. Lacey is saying that these kind of engineering practices are important for the Scrum teams but their absence is not as critical on the four-week Sprint compared to the one-week Sprint. The current team practices are supporting for each of the developers working on own tasks and

they rarely share these tasks between. For this reason, the teams are not yet very good on decomposing the tasks and Lacey is recommending for these kinds of teams to use from the two to the four-week Sprints. (Lacey 2015, Chapter 6.)

One more issue should be considered on determining the Sprint length. Currently, the organization's release cycle is a one month. To not require to change this, obviously the four-week Sprint would best choice. The two-week Sprint and the one-week Sprint could still have a little advantage as this approach would offer possibility for the customers to review the developed solutions earlier and get a feedback to make required changes before an official release date.

Considering all facts above, the two-week Sprint length would be most recommended to be started with the Scrum.

5.7 Handling maintenance in Scrum

When taking the Scrum in use, there should not be implement any special approaches for maintenance, such as for example own backlogs for maintenance issues and how the Scrum team to handles incoming bugs compared to feature development. Incoming bugs should be handled as same as any other work is handled. Those are added on the product backlog and prioritized by the Product Owner. Reasons for this is that when looking this topic from product's users point of view, they are not caring difference between if an issue existing is required new feature or is it the bug, they are just expecting a system to be working differently. (Cohn 2009, refereed 02.04.2017.)

How the teams should handle incoming critical issues, there are some facts to be considered. The Scrum's rules are stating that when the Sprint goal is set and the Sprint execution is started, there should not be made any changes that could essentially affect the Sprint goal. This rule is expected by the Product Owner to follow as if she or he changes a developed feature during the Sprint to another feature, this decision will produce the waste as all work related to the previous feature might have to be thrown away and even it may generate a more work for needing to remove the previous feature away first as a partially completed work should not be included in a potentially shippable product increment at the end of the Sprint. This kind of decisions will also affect trust between the Product Owner and the Scrum team, because the team cannot trust the Product Owner for keeping commitments. Although this is still a rule and not a law, and when the critical

issue occurs on the product and the current Scrum team is the only one who can fix it, a practical approach should be used. There will be consequences described above for making changes on the Sprint goal but if consequences are far less than consequences of deferring changes such as very critical production bugs, making change is a smart business decision. (Kenneth 2012, 70-72.)

6 CONCLUSION

To achieving the set goal to move the organization towards using the Agile and the Lean approaches on their work, and use the Scrum practice, requires changes on the people's behaviors and skills, and the roles on the organization. The organization needs also understanding what would be good length for the Sprint and how to handle the maintenance in Scrum.

To achieve one of the very key things, the self-organizing teams, requires changes for everyone. For the managers to support it, they need to change their behavior away from the command and control. They role needs to be more on only setting goals and boundaries for the teams, and then delegating power for the teams by giving them freedom to achieve the set goals. This practice does not mean that the managers leave the teams face their challenges alone but the managers are required to be responsive to support the teams when needed and use coaching as one of the key practices for supporting them. The whole organization needs to make sure that the continuous improvement practices are taken in to use and as the Scrum offers two approaches, the Sprint Review and the Sprint Retrospective, these are where focus should be set so that the whole organization understand those values and supports them. For final aspect to achieve the cross-functional teams, the Developers are required to learn new technology skills from. the NET/C# technology and the web technologies

.

When taking the Scrum in to use, the Developers and the Quality Analysts are required to change their behavior. They need to understand that there is not an individual work anymore but responsibility from all of the work belongs to the whole team. The Developers need to start working more on same features and here can be utilized practices from the Kanban to set limitations how many items the team can work on same time and also utilize the pair programming practice. The team needs to start understanding that the Quality Analyst are part of the Scrum team and they are required to take on the feature planning and the estimation process work as without this the team is not using all of the skills and the knowledge what they have. The team needs to understand that anymore the Quality Analyst is not responsible alone for a quality what the team produces as it is whole team responsible and a quality related task can be completed by all of the team members. In the Agile, team practices are moving more talking about requirements then writing them down and the Quality Analysts are required to change to behavior to talk more about with the Product Owner and different stakeholders how a feature should work and not expecting anymore just a

written documentation. They also need start learning how to work iteratively so that they do not end up in a situation where they produce lengthy delays on the Sprints when waiting work coming from the Developers to the Quality Analysts.

The Scrum team members are required together to define the Definition of Done before taken the Scrum in use. This way they would share same understanding what it means when a work is really done, will make them feel more as the team when they are together committing it and will remove a guesswork from iterations. This would also provide team's stakeholders and others understanding what it means when the teams are saying that their work is done.

There are two aspects relating on role changes: Who should become the Scrum Master and how to solve challenge for four Product Owners vs three teams. When organization appoints the Scrum Masters for the teams, it should not go automatically on the way that the current Team Leaders will come to the Scrum Master. There is need to have understanding what the Scrum Master role is before making any choices as the role it not same as a team's technical lead like the Team Leader's role has been now. Choosing should made through looking is a person willing to take the role and also does the person have skills and characteristics needed on the Scrum Master role.

Situation having four Product Owners and three Development Teams needs to be changed when implementing the Scrum, and also practice how work is assigned from the Product Owners to other teams then their own. There is required to analyze the Product Owners responsible areas so that they in the future assign the work only on their own team's backlog. One of the Product Owner is required to change to start working on a Developer's role and as there is already been one of the Product Owners doing Developer's work sometimes, this is the target person for who's role should be changed.

To implement the Scrum, there is need for the organization to understand what would be a good length for use as a Sprint length and how they should handle a maintenance with the Scrum. For the Sprint length is recommended to take a two weeks length. This recommendation based on facts that that any feature work usually last less than three months so with the two weeks there will be enough possibility to get a feedback compared to three or four weeks. Making the Sprint length only for a one week would stress customers as they are not familiar with the Scrum and the Sprint model. Also using a one-week Sprint would require the teams to be experienced in the Scrum, their team engineering practices would need to be in a good level and the team should have a good

ability for decomposing a work. The two-week Sprint practice also fit on a current one-month release cycle model and would not require to make changes on it.

For handling a maintenance there should not be build any special approaches. Maintenance issues should be handled on a same way as other backlog items by adding them on team's backlog. When new maintenance issues are coming on the team, they should not be taken on an ongoing Sprint as it would lead affecting a waste when the team are required to change their commitments for an ongoing work. For critical issues, there is still recommended to use a practical approach. If the maintenance issue is for example a very critical production bug, then consequences are far less what happens when a Sprint Goal is changed comparing to consequences what happens if bug is not taken in the Sprint.

7 DISCUSSION

The company's high management have set the expectation for these changes and the new practices. These expectations were to produce a more value for the end customers by speeding up the product development, make it more responsive to the new customer requirements, increasing the software quality and the customer satisfaction. All of these expectations are such, which can be seen to exist inside the Agile and the Lean values and as all of the defined changes and the new practices are based on theories behind both of these, it is very expected that improvements will happen on R&D and Product Management.

These changes are requiring to change people behaviors and only here defined results are not enough take people through these changes easily. There is highly recommended to define a change management process how people are taken through these changes. This way it would provide much more effective and faster way to take here defined results into use.

This thesis results are directly defined to the R&D Development Team members, the Product Owners and their managers but there will be also indirectly involved several other people too. The Fast Response Unit member and other than QA Analyst in the RMD will be quite heavily involved on these changes and new practices, and naturally other departments such as the Support and the Sales will be involved too. It would be expected than when these results are taken into use, it will require to start analyses to define changes also in these other teams and departments. The customers will be also seeing changes towards them and to start this path with them, it would be recommended to define process how the customers are taught about the Agile and the Lean. In this way, it would prevent happening a sub-optimization on the organization as there would be looked the whole value stream.

These results were the ones, which was seen the most required ones. On theory side, there would have been possible to define several other new changes and new practices but this would have not follow anymore idea of the iterative approach, one of the core ideas in Agile. More better approach is to start with these and during implementing these, there could be then started to define the new changes and the new practices, and use learnings from these first ones to define them.

Mostly all of these results are not unique for this R&D department or even for this company. As there are quite many other R&D departments inside the company, using this thesis results would be beneficial for them. Even if some of them are using Scrum approach already, they could get benefits from some of the results, for example when they are looking for the person to be Scrum Master. Other companies could also use these results as development practices are not that different between software companies. Especially if there are plans to move towards the Agile and Lean from the Waterfall-based approach, this would give them a good starting point to define own required changes and new practices.

REFERENCES

Adkins, L. 2010. Coaching agile teams: a companion for ScrumMasters, agile coaches, and project managers in transition. United States of America: Pearson Education, Inc.

Agile Manifesto 2001. Refereed 16.10.2016, http://www.agilemanifesto.org.

Appelo, J. 2010. Management 3.0: leading Agile developers, developing Agile leaders. United States of America: Pearson Education, Inc.

Ashmore, S. & Runyan, K. 2014. Introduction to agile methods. United States of America: Addison-Wesley Professional.

Cohn, M. 2009. Succeeding with Agile. United States of America: Pearson Education, Inc.

Cohn, M. 2009. Bugs on the Product Backlog. Mountain Goat Software. Refereed 02.04.2017, https://www.mountaingoatsoftware.com/blog/bugs-on-the-product-backlog.

Crookshanks, E. 2014. Practical Software Development Techniques: Tools and Techniques for Building Enterprise Software. United States of America: Apress.

Benignton, H. 1983. Production of Large Computer Programs.

Crispin, L & Gregory, J. 2009, Agile testing: a practical guide for testers and agile teams. United States of America: Pearson Education, Inc.

Greene, J & Stellman, A. 2014. Learning agile. United States of America: O'Reilly Media, Inc.

Davis, B. 2013. Agile practices for Waterfall Projects. United States of America: J. Ross Publishing.

Hammaberg, M & Sundén, J. 2014. Kanban in Action. United States of America: Manning Publications Co.

Howard, K & Rogers, B. 2011. Individual and Interactions: An Agile Guide. United States of America: Pearson Education.

Kenneth, S. Rubin. 2012. Essential Scrum. United States of America: Addison-Wesley.

Kniberg, H. 2007. Scrum and XP from the Trenches. How we do Scrum. 2007. United States of America: C4Media Inc.

Kniberg, H. & Skarin, M. 2010. Kanban and Scrum – making most of both. United States of America: C4Media Inc.

Kniberg, H. 2011. Lean from the Trenches - Managing Large-Scale Projects with Kanban. United States of America: The Pragmatic Programmers, LLC.

Lacey, M. 2015. The Scrum Field Guide: Agile Advice for Your First Year and Beyond, Second Edition. United States of America: Addison-Wesley.

Larman, C. & Basili, V. 2003. Iterative and incremental developments. a brief history.Computer, 36(6), 47-56.

Measey, P & Radtac. 2015. Agile Foundations. Principles, practices and frameworks. Swindon: BCS Learning & Development Limited.

Medinilla, Ángel. 2012. Agile Management, Leadership in an Agile Environment. Germany: Springer-Verlag.

Principles behind the Agile Manifesto 2001. Refereed 07.02.2016, http://www.agilemanifesto.org/principles.html

Poppendieck, M. & Poppendieck, T. 2003. Lean Software Development: An Agile Toolkit. United States of America: Addison Wesley.

Poppendieck, M. & Poppendieck, T. 2006. Implementing lean software development: from concept to cash. United States of America: Addison Wesley Professional.

Royce, W. 1970. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON 26. United States of America.

Schwaber, K & Sutherland, J. 2016. The Scrum Guide. Scrum.Org and ScrumInc.

Takeuchi, H & Nonaka, I. 1986. The New Product Development Game. Harward Business Review. Refereed 09.10.2016, https://hbr.org/1986/01/the-new-new-product-development-game.

Williams, L & Kessler, R. 2002. Pair Programming Illuminated. United States of America: Pearson Education, Inc.