OULUN AMMATTIKORKEAKOULU

Jani Kukkohovi

AGILE DEVELOPMENT AND TESTING IN EMBEDDED SYSTEMS

AGILE DEVELOPMENT AND TESTING IN EMBEDDED SYSTEMS

Jani Kukkohovi
Master's thesis
Spring 2014
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Jani Kukkohovi
Title of thesis: Agile Development and Testing in Embedded Systems
Supervisor: Markku Rahikainen
Term and year of completion: Spring 2014 Pages: 49 + 1 appendix

---

It is fair to say that Agile development is today's major trend. It is used more or less in every company if the company is involved in mobile software industry. Agile development is also widely used in other embedded software development. It is recognized to be a very effective and efficient way of development. However, there are many issues which are needed to be considered and taken into account when transferring to this process model. Especially the transferring period can be very difficult and have a long lasting effect to a company.

The aim of this Master's thesis was to study more deeply this subject and find a different way to conquer the challenges what comes to this subject. There are so many positive experiences from this, but normally some of the negative sides have been left out. I have experienced a transfer in my own career and I saw really close what kind of things are usually left out and not told.

The second aim of this thesis was to investigate how well Agile methods suit in embedded software development. Agile works well what comes just to software projects, but when talking about embedded development, the process is not always so easy. Embedded devices vary for example from small music players to big and complex medical devices. This thesis will concentrate only on small portable devices, since my experience comes from that field.

I used mainly my own experiences and situations I have met to find a suitable process. In theory parts I used different publications from the Internet and books to back up my knowledge. I managed to present basic practices and processes from different methodologies that I discovered as good in embedded development. Next steps would be to test these different processes in practice. Since there are many different kinds of embedded systems out there, every modified process has to be first tested in a real life before it can be decided whether can it be taken in to use or not.

Keywords: Agile, Agile development, Scrum, embedded development, feature driven development, test driven development, Agile testing

CONTENTS

1 INTRODUCTION

Since 2001, when Agile manifesto was declared, Agile software development has grown in big steps becoming more and more used around the world. General feeling has been for a long time that Agile is not necessarily very suitable for an embedded software development. There have been lots of studies and researches about this topic and results have been positive. However, the truth behind this and what comes to my experiences is that the story is not always so nice to tell. There are many obstacles and points needed to be taken in to account to make the project work smoothly and effectively.

Before the manifesto was declared, projects were having problems to keep their schedules. Budgets were exceeded and business needs were not always archived. In other words software development had big problems as a working development process. When talking about an embedded software development, things were even worse. In the embedded software development some of these problems are still valid. Projects have become more and more complex which makes it much more difficult to find a suitable process. This was one of the biggest things that motivated me to this thesis work.

This thesis is not just concentrated on one Agile method. The target is to investigate many different methods and see if some custom method could be used to meet all expectations and needs what comes to Agile development in embedded software projects. The reason for this kind of starting point is that there have been lots of investigations using just one particular process model. Due to my experience this has not worked in the projects I have taken part in.

Finally, I try to present a working process which could be used in the embedded software development. While creating a new process, I try to evaluate how the new process would respond to common problems in the embedded software development.

2 PLAN DRIVEN SOFTWARE DEVELOPMENT PROCESSES

2.1 Waterfall model

Winston W. Royce was the first one who formally described a Waterfall model in 1970 [1, p. 328-338], even though Royce did not use the word Waterfall. The Waterfall model is useful to point out to developers what they need to do. The model is a sequential design process, which is used in software development processes where progress is seen flowing steadily downwards like a Waterfall. In figure 1 the Waterfall process model description is shown. It is wise to use this model when a customer knows exactly the requirements, they are well defined, and are able to wait for the system to be ready for release. This means that the model faces problems if requirements are changing.

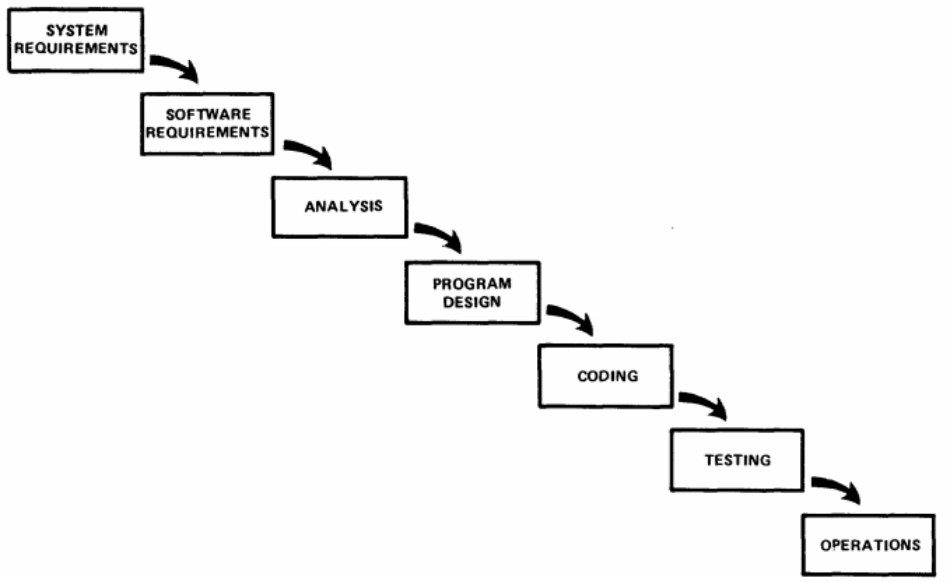

FIGURE 1. Waterfall model [1, p.329]

2.1.1 Known problems of Waterfall

After finishing each phase, this model moves to the next one. If reviews occur before moving to the next phase, it might raise reasons to make changes. Sometimes reviews are also held to ensure that the phase is completed. Therefore, this model does not courage revisiting any earlier phases once it has been completed. This is the main reason that this model is not commonly thought as a flexible project model and has received lots of criticism. The Waterfall model is also very slow what comes to changes.

What comes to the embedded system development the Waterfall is not a very realistic method, since bugs are often found in lasts phases and the bugs should have been corrected already in an earlier phase. There might be some input condition which has been forgotten to handle in a proper way. To verify and complete the desired behavior, a prototype is often needed.

2.2 V-model

The name of V-model comes from a verification and validation process. Some modifications have been done to this model, but the first one (V-model 97) was a software development standard for IT projects by the German government. Like in the Waterfall model, a process cannot move backwards in phases. This sets same preconditions for an easy and successful completion; requirements must be known and frozen. In a well structured and controlled development project it is a good basis. In this model all acceptance tests and design elements must be traceable towards one or more of the system requirements. Figure 2 shows the V-process model description [2].

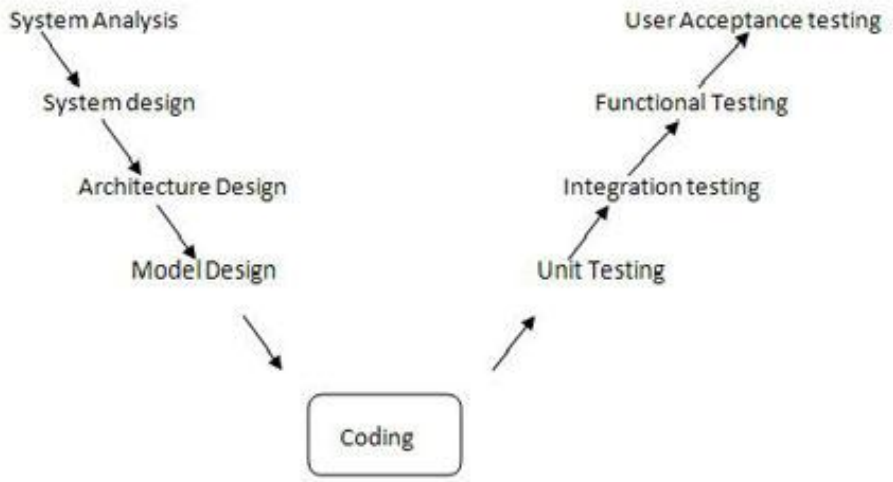

FIGURE 2. V-model [2]

2.2.1 Known problems of V-model

When inspecting the picture, it is easy to notice that in the V-model software is defined and designed on the left and built and tested on the right. Unlike in iterative process models, testing and defects are found much later in the V-model. Often a smoke testing is used as a testing type in early stages allowing to find out what the system is all about.

The testing in the V-model is the same kind of gate keeping as in the Waterfall. If the software works like it are required, specified and designed, it can be released. Overall the V-model is a slower process than Scrum. Like the Waterfall-model, the V-model is best fitted to projects where the requirements do not change.

3 AGILE SOFTWARE DEVELOPMENT PROCESSES

This chapter briefly describes the main points of the Agile development and inspects why it has become so popular since the Agile Manifesto was declared 2001. There are many different methodologies, but in this thesis only the most used are inspected. First, there are some common details about the Agile development. Then some most used methods: Scrum, a feature driven development, a test driven development and an extreme programming. At the end there are some problems and difficulties presented what comes to Agile methods and especially Scrum. The reason I write especially about Scrum is that it has been mainly used by my previous employers.

3.1 Agile Development

When Agile was born, there were 17 software developers with different backgrounds in a meeting brainstorming ideas about a software development. An interesting part here is that all those different people were able to agree on terms and principles how a software development should be done and which direction it should be driven towards. The group named their self "Agile Alliance". The result from this was Agile manifesto (Appendix 1).

An Agile development is an umbrella term for multiple incremental and iterative software development methodologies. Even if every methodology is unique, they all share the same vision and basic values (the Agile Manifesto). They all work in a same iterative way, which bases on a continuous feedback and making a solid software releasing system. All methodologies include a continuous evolution, meaning a continuous testing, continuous planning and continuous integration. They are all light-weight processes, especially if compared to tradi-

tional Waterfall-based processes. Finally, but not least they all empower people to co-operate and make quick decisions together fast and effectively.

The traditional software development backs on the Waterfall-model where there are tight procedures for work and tight requirements for delivered artifacts. Even so, many of these software projects fail.

Pareto's law is used in many situations, but it also suits well to the Agile development. Pareto's law is also known as an 80/20 principle. “80 percent of your results come from 20% of your efforts" [27]. Even it is very difficult, we should try to find that most important 20% which brings majority of the results. It does not mean that software is usually bad, just that some features bring more results and some features are not necessarily worth of spending resources that much. This law is really good and worth to think about, but seeing what is "that 20%" is just often impossible.

The Agile development suits best for small and medium sized projects. However there are already some results gained and available showing that Agile has been used successfully also in bigger projects. A general thought is that Agile does not fit well to the embedded software development. Embedded software development is therefore challenging to Agile, since the development process is so dependent on hardware. Often the final hardware version is ready and available in the very final stages of the project. This thesis tries to find a solution for the most common problems that have been encountered in the embedded software development in the Agile model and tries to present a working process which suits well for also larger embedded software projects.

A Standish group has been collecting information about a real-life IT environment and software projects since 1985. The Standish group published their first CHAOS report (also called a CHAOS manifesto) in 1994. In this report there are the results of investigation about software projects and how they are completed, failed or challenged. The report offers a lot of interesting data about how a soft-

ware project's results have changed during years. In figure 3 there is a pie chart about the successful rate of software project.

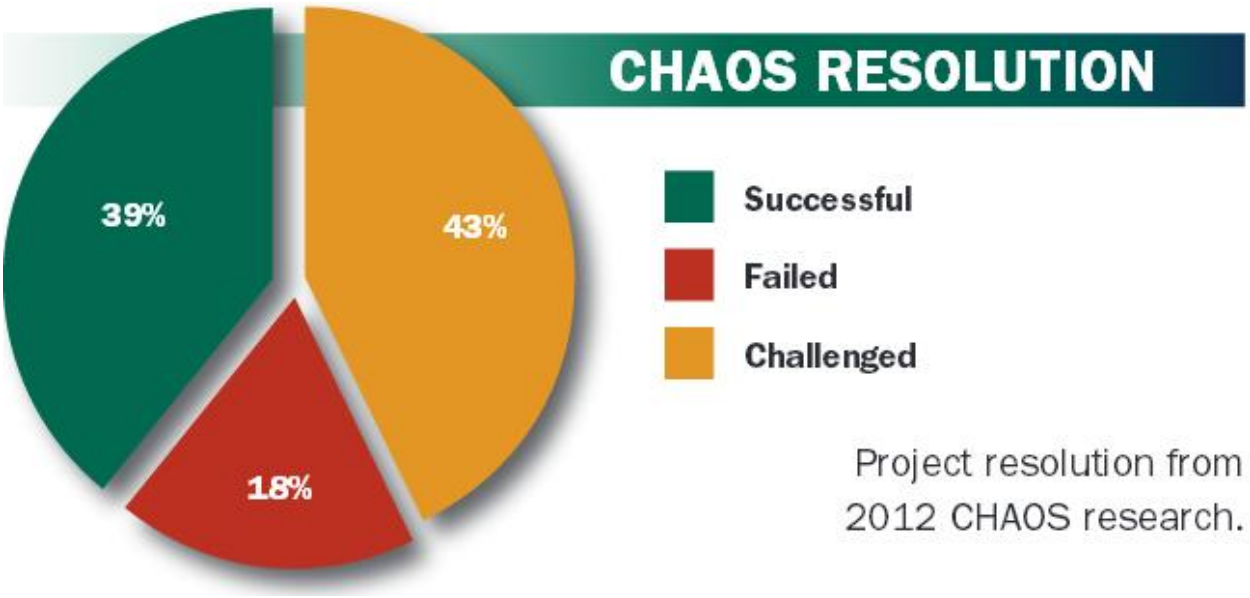

FIGURE 3. Chaos resolution in 2012 [16, p.1]

In the CHAOS research 60% of projects were located in USA, 25% were from Europe and last 15% from the rest of the world. The database has approximately 50,000 projects in it. In figure 4 the CHAOS results from past years are presented.

RESOLUTION

	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

FIGURE 4. CHAOS results between 2004 and 2012 [16, p. 1]

There have also been older projects in the database, but a new database removed projects from 1994 to 2002 since they did not match the requirements so that a analysis could be properly done. Over 40,000 projects were removed in this clean up.

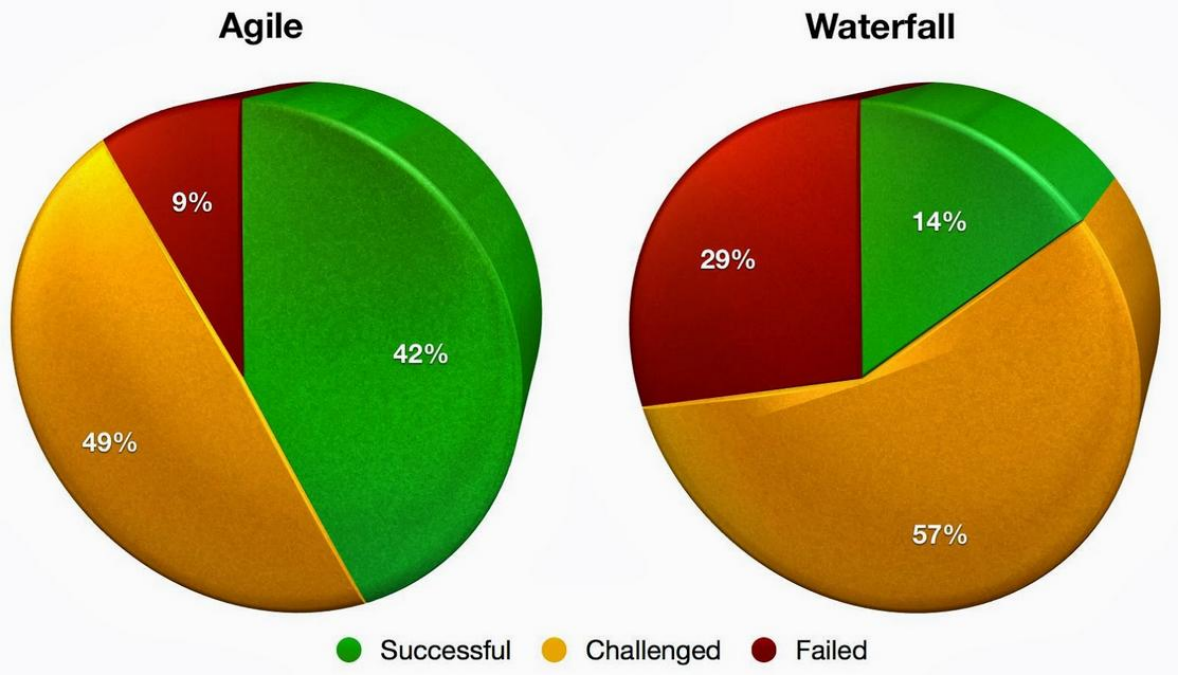

FIGURE 5. Agile and Waterfall comparison 2002-2010 [17, p. 25].

In figure 5 successful means that software with required features and functions are delivered on time and inside the budget. Challenge means that software is delivered late, with less feature or/and features than required and/or over the budget. Cancelled means that software was never delivered due to project cancellation.

In 2002 less than 2% of overall projects and less than 5% of new application development projects were using Agile methods. In year 2011 the same percentages were 9% of all projects and 29% of new application development projects. [17, p. 25]

In the same Chaos report there is a mention about the quality of Agile projects: “The Agile process is delivering not only a higher percentage of features driving up the average, but also a higher percentage of higher usage of those features. Still, there is much need for improvement.”

All Agile methods use the Agile manifesto as a guideline of doing. There are four key values and those values are based upon twelve Agile manifesto principles.

Value 1 - Individuals and interactions over process and tools: The Agile method's big note to a team way of working is a face to face communication. Therefore, it is a big advantage that the whole team operates on the same site. Also, a customer's onsite presence is valuable. There are multiple positive outcomes in this co-located team. The face-to-face communication is much more efficient than e.g. emails. The teamwork is stronger and there are more chances for innovations. All this leads to a better job satisfaction.

Value 2 - Working software over a comprehensive documentation: The documentation is reflected to working software. Boehm and Turner [18] describe the Agile process motto in a funny but effective way as YAGNI which means "You aren't gonna need it". This motto means that limit documentation and design to a just required level. The highest priority, what a customer values, is to meet incremental targets and requirements. A product is developed on time and delivered to a customer. Lessons learned are held and feedback from a customer received after each iteration.

Value 3 - A customer collaboration over a contract negotiation: A close and tight communication with a customer is kept. This helps to deliver a desired kind of product to a customer. It affects also to a product quality by reducing defects.

Value 4 - Responding to a change over following a plan: By nature the Agile methods are not that much of a predictive kind, rather adaptive. At the moment the business world is very dynamic. Requirements might change in a very fast term, therefore it is very important to be able to adapt to new requirements and leave an earlier defined plan a side.

The Agile way of working requires more attention, feedback and co-operation from a customer than a traditional development model. For the customer it is not necessarily easy to explain that more commitment and interaction is needed. It is promised by Agile that customers will be brought closer to developers and this will help them to solve many general problems, e.g. how to deliver best from what is needed and how to handle problems now and in the future. Getting customers more involved is not easy since usually customers are used to just give the requirements and to make a contract with a fixed price. A customer might not be eager to spend more time to the software development. Also, this regular face-to-face communication might make some developers uncomfortable, since not every developer is used to this. Maintaining simplicity is not always easy and therefore sometimes it might cause a lot of extra work to keep that. The Agile model lifecycle is presented in figure 6.

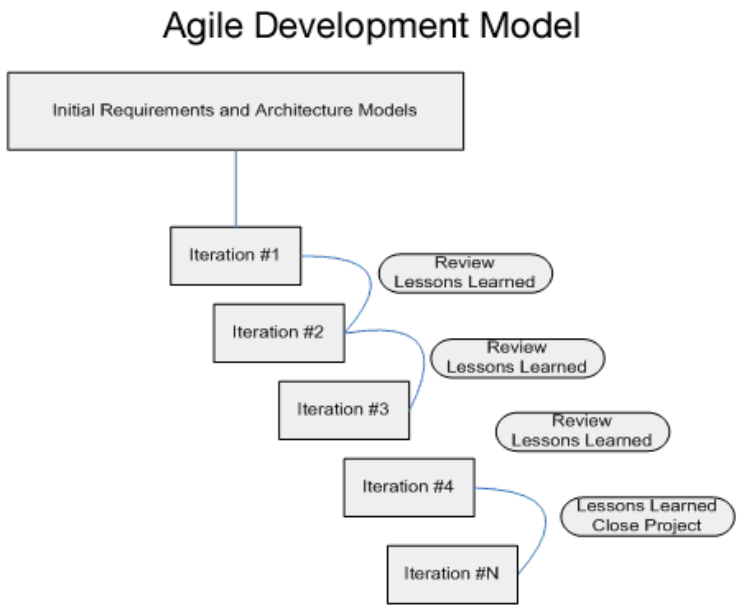

FIGURE 6. Agile model lifecycle [19, p. 29].

In figure 7 it is shown what Jim Johnson, the chairman of the Standish group international, claimed: only 20% of features are often or always used in software development projects. This is one of the biggest, if not the biggest waste in the software development.

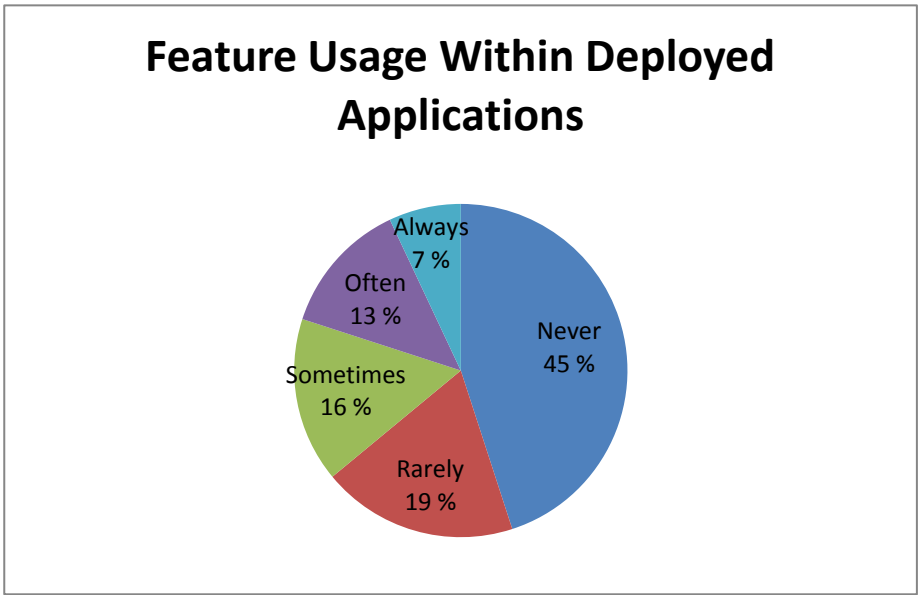

FIGURE 7. Feature usage within deployes applications [21].

Switching between tasks is usually seen bad and not recommended, therefore, many methodologies recommend to getting one task done at the time. After all, the Agile development has been quite efficient and it has been improved all the time by many software development teams. The increased productivity in a team has brought predictiviness to their doing. Normal difficulties in the software development are multiplied when speaking of the embedded software development due to an indirect and constrained environment. As an outcome of this, embedded developers are often more skilled and disciplined being more aligned with engineering than programming. All problems usually lead to another problem, therefore, it would be important to make every detail right. For example, an unpredictable delivery leads to a pressure from scheduling and non-realistic plans. The pressure from scheduling leads to short cuts and long hours when facing problems. The Short cuts lead to defects and the defects increase the amount of long hours. The long hours lead to burn out. In figure 8 there are gathered the problems that are confronted in a software development process. [24, p. 2]

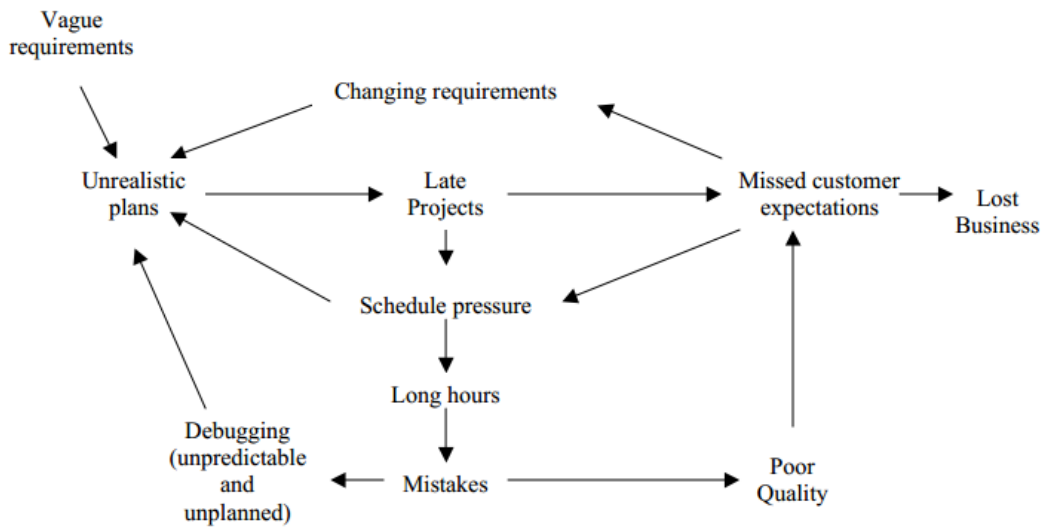

FIGURE 8. Vicious cycle [24, p.2]

3.2 Scrum

Scrum is a lightweight Agile process meant to be used to control and manage a software development embracing iterative and incremental practices. The first time Scrum was introduced in 1986 by Takeuchi and Nonaka [3]. Later in 1995 this was refined by the same people. In 2001 the Scrum process was fully described by Ken Schwaber and Mike Beedle [10]. Scrum concentrates on what is really important: managing a project or writing software that produces a business value, therefore, requiring only very few artifacts. In figure 9 a standard Scrum process is presented, including artifacts, processes and members.

FIGURE 9. Standard Scrum process [11].

Scrum uses three different types of roles: Product owner, Scrum master and team member. Ideally the team size should be between 5 and 10 members. The team itself should be cross-functional having members from different areas like QA, development and UI designing. The team works in from 1 to 4 weeks sprints and after each sprint a shippable delivery is released with the features that were selected. These sprints are repeated so many times that a product backlog is empty. The sprint backlog is planned before each sprint starts. This planning is done by all members. The sprint backlog comprises product backlog items which the team think that they can finalize during the next sprint. A burn down chart is very informative and widely used metric to present how much work is to be done compared to the time left in the sprint. In figure 10 a typical Scrum burn down chart is presented. In figure 11 a Scrum process is presented, having a 30-day-long sprint.

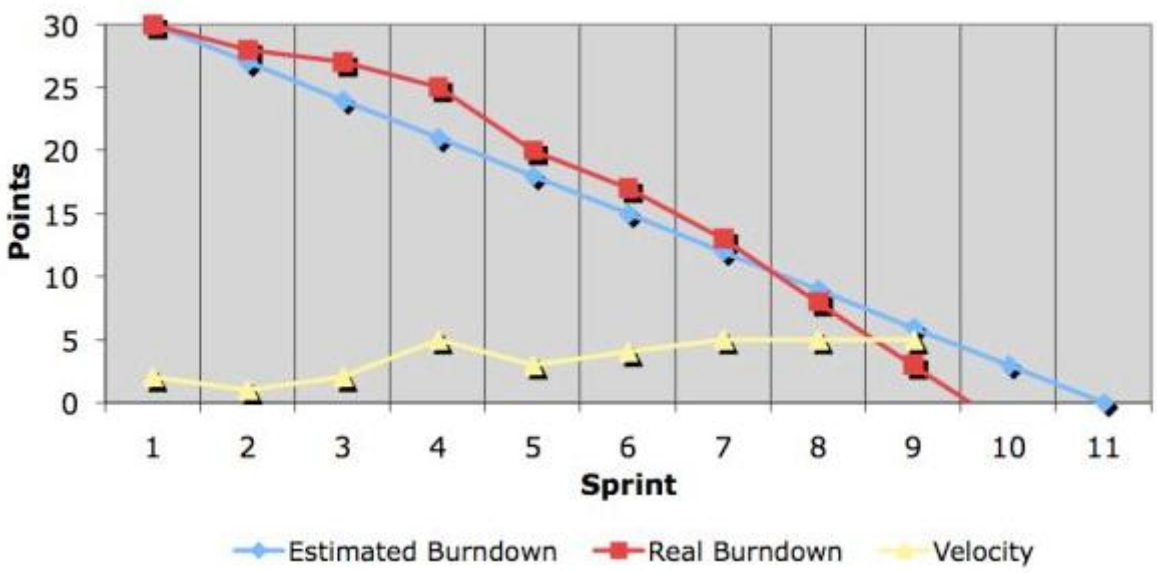

FIGURE 10. Scrum burn down chart

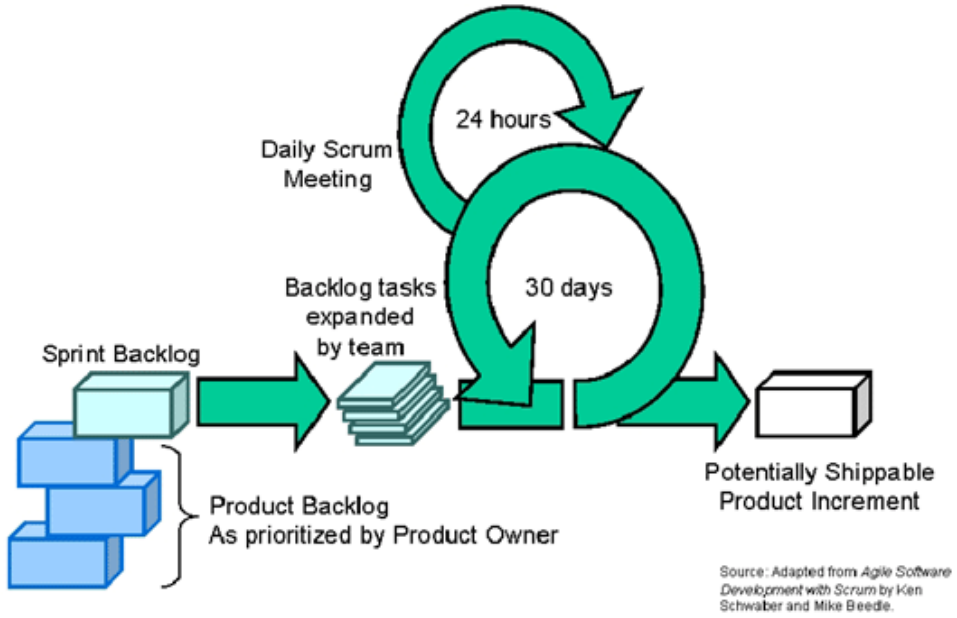

FIGURE 11. Scrum process [9]

3.3 Feature driven development

A feature driven development is also a incremental model of a driven and short iterative process. The process was developed by Jeff De Luca for a relatively large software project working in the banking industry in Singapore 1997. The first original process was heavily affected by Peter Coad's thoughts of development processes, object modeling and color modeling. At the beginning the feature driven development became one of the most used Agile software development methods, but later on it has been partially replaced by other models like Scrum.

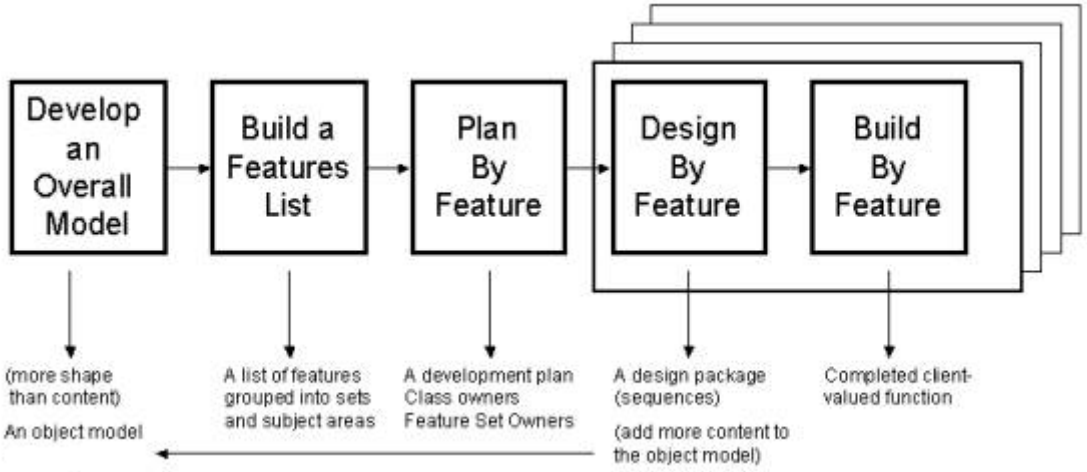

FIGURE 12. Feature driven development process [12]

The Feature driven development consists of five basic activities [13, p. 106-181].

- Develop an overall model
- Build a feature list
- Plan by feature
- Design by feature
- Build by feature

Develop an overall model

The first official step in this model is to make an overall high-level chart of the whole system and its context. There could be some prototyping or business planning done before this step. These activities are divided into three different phases. The first three activities cover the first phase, design by feature being the second phase and build by feature being the last phase.

Build a feature list

The feature driven development uses features to communicate about requirements with the customer. In this point a project is also divided into smaller parts which are easy to implement. A set of features is first started by building a feature list. These features are dealt to subject areas. As a result of this phase, there is a list of subject areas consisting business activities and to complete them also feature matching activities.

Plan by feature

Probably the most important part of the first phase is to plan a developing schedule to implement features. A chief developer is responsible for the feature set assigning classes to developers. The first phase ends the design and analysis phase and is followed by lessons learned, where it is studied how well the phase has succeeded.

Design by feature

In this second phase the chief developer selects a suitable group of features that can and will be developed within next the iteration. The developing team together with the chief developer creates a sequence model for each feature.

Build by feature

In the final phase the developing team starts developing the designed features. After developing is done, the developers start to write unit tests to gather as much information as possible of the quality of their work. Usually unit tests are written by other developer as long as both developers agree with the functionality and design. Right after classes have been run by unit tests without any failures, the classes are sent to the chief developer to integrate classes to a feature and to make a build out of feature.

It is easy to see that the chief programmer's position and role is the key to the success in this model. The chief developer holds much responsibility and is critical for the project. The chief programmer can be a lead developer or similar who has a strong experience and technical status. The Feature driven development has proved itself effective in some projects that have needed to be rescued from a delay or a complete fail.

3.4 Test Driven Development

One of the core practices of extreme programming is a test driven development. The Test driven development was first practiced in an extreme programming. However the test driven development can be implemented in any software development methodology.

Compared to the traditional development, the test driven development turns it more or less upside down. It requires a developer to write a code and an automated test code simultaneously. The traditional development style might lead to a not needed code being implemented since there is no direct mapping between requirements and code. Sometimes it can also lead to a situation, where some requirements are not being implemented in the code. The test driven development is based on a simple rule that no functionality is implemented and added without a test. A feature which does not have a covering test is not added or tested towards if it is written first. In other words the test driven development provides means of direct mapping between the requirements and code. When checking a bug fix, a test is again added to make sure that the fix is working properly. This cuts down unnecessary rounds between testing and development.

The first step is to write test a code which makes a code to fail. It does not need to be complex just that the code fails. The test driven development style forces developers to go through and think about the requirements before writing any code. This might be difficult to be mentally adjusted to by developers, but various benefits and advantages will be easily seen after it has been done. In figure 13 the test driven development process is shown.

A big challenge to a company or a team is to find the right tools and techniques when turning to use the Agile methods and test driven development. Even harder it is when trying to do that without compromising the already gained efficiency with self made custom tools. [5, p.43-50]

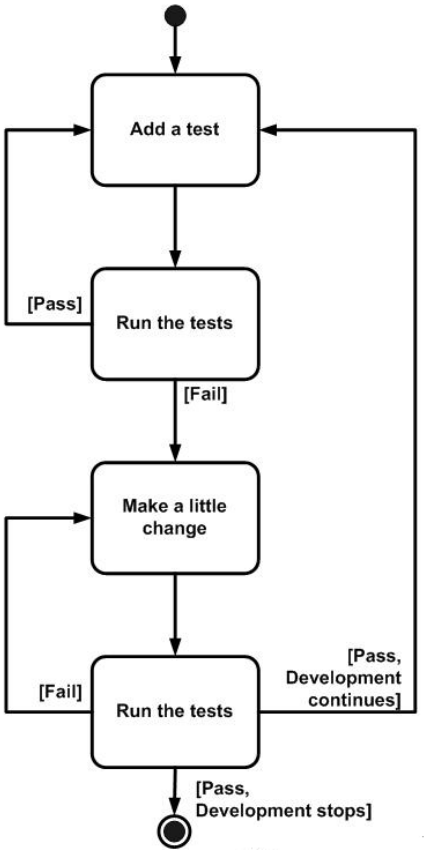

FIGURE 13. Test driven development process [15]

3.5 Extreme programming

Since the first extreme programming project was started in 1996 it has become one of the several popular Agile processes. From the very beginning it proved to be very successful at many companies around the world. The extreme programming has become successful because it emphasizes customer satisfaction. Instead of having one particular day far away in the future when delivering everything, you could possibly want this process to deliver the software you need when you need it. The extreme programming highlights the team work. Everyone, no matter in which position you are, is an equal partner in a collaborative team. The extreme programming has indicated to improve a software project in five different ways:

- communication
- simplicity
- feedback
- respect
- courage

In this process developers are having a lot of responsibility, even more than in some other Agile processes. The developers constantly communicate with other developers as well as with their customer. The developers keep their design clean and simple. There is a small bunch of simple rules what comes to the extreme programming. The rules may seem to be like a jig saw puzzle: many small pieces which make no sense on their own, but when all are joint together, the complete picture comes clear. In figure 14 it is shown how rules work together. [7] In figure 15 an example of how the iteration goes in the extreme programming is shown.

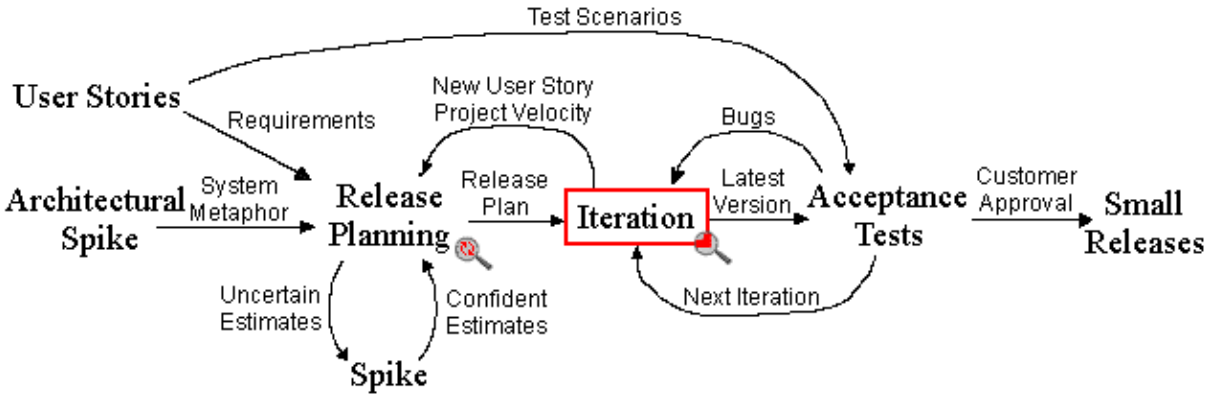

FIGURE 14. Extreme programming flow chart. [7]

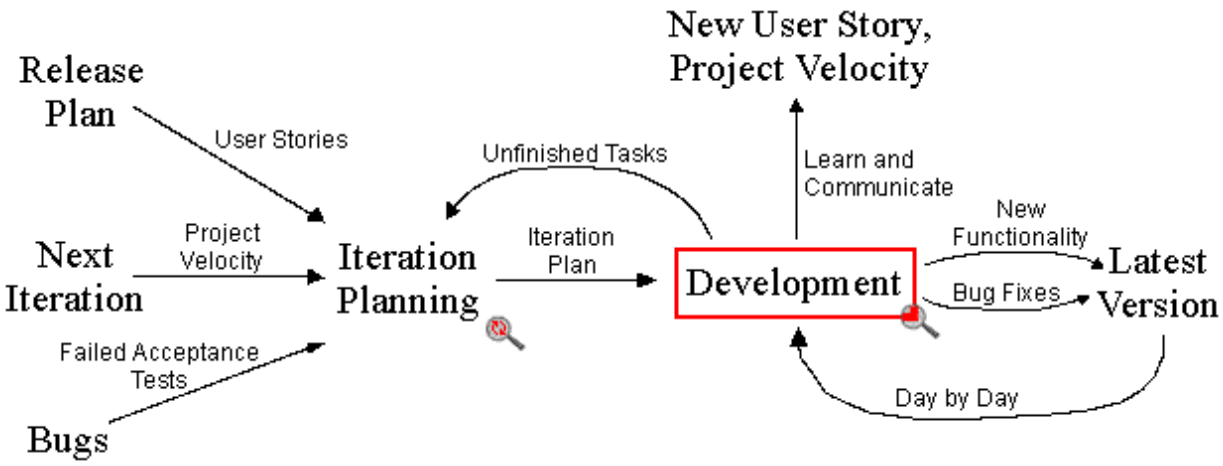

FIGURE 15. Extreme programming iteration. [7]

Since the extreme programming requires a lot of discipline and responsibility from a developer, it has raised a general thought that this works well only with senior level developers. From the testing point this is also challenging. In the extreme programming all codes must have unit tests and all codes must pass all unit tests before it can be released. However, often these tasks are done by the developers [7].

3.6 Known problems of Scrum

The Agile methods use very short iterations, usually from 1 to 4 weeks. This means that the working software is released quite often to the stakeholders so that they can check if the software is what they need and how it should be. The stakeholders can make changes to the requirement(s) and make a prioritization to the features that are seen valuable to a customer.

Software has to be tested after every iteration. In Scrum and other iterative models there are broken dependencies, a low test coverage and a lack of compliance to coding standards. The lack of compliance to coding standards is due to the fact that there is not much time to do the implementation in one iteration.

Every sprint is planned in the planning session. Since sprints are very short and there is not much time to do all tasks scheduled to that certain sprint, it is very hard to find time to do this planning session. Generally, it is presented that the planning is done in a "short planning session". To keep that planning session short, it needs a huge amount of experience and professional skills to make that happen since it is literally predicting future.

What comes to testing in the Agile process, it is best to use an exploratory testing because it does not need that much preparation. Important and most crucial bugs are found quickly towards the short execution time period. The exploratory testing also does not need that much planning to get started. After the first spring, it might be challenging if there is not good enough code to be tested. Also, if lots of bugs are being reported during the sprint, when is it time to correct all of them? Other good and important testing type is a regression testing. This is very important since it ensures the existing functionality so that it does not break up with a new implementation or bug fixes. The Scrum development works well when a team is located in one and same place. However, nowadays it has become more and more usual that the teams are functioning in more than one place. This is a major hindrance to day-to-day tasks.

From the Scrum master point of view Scrum meetings are sometimes also seen as a negative thing. Keeping approximately a 15-minute meeting each day feels sometimes not needed. Why to have a meeting where everyone talks about things that are already known? This means a situation when you are working with a backlog. If you are not working with the backlog, a 15-minute meeting is not enough to get in to the level that is needed. The final result is that either you have a 15-minute meeting where everybody repeats the things that are already known or you keep a 1-hour meeting where everybody talks about technical details, issues and problems they are having.

4 UNIFIED PROCESS

A Unified process is a use case driven, iterative and incremental development process framework. Including different type of software systems like a small-scale system and large-scale one, the unified process is applicable even when having many levels of managerial and technical complexity. The Unified process is a framework providing an infrastructure for executing projects without needing all the details which are required for executing projects. Most importantly it is a software development process framework, a model for project's life-cycle including collaborations, iterations and context. The creators of unified process realized that it was nearly impossible to specify all possible requirements before moving to the next phases, analysis and design. Therefore, each phase must be able to interact with the previous phases. The Unified process is generally divided into four different phases:

Inception: In the inception phase a project scope and business case are solved. Feasibility is also investigated, use cases are being defined and requirements gathered.

Elaboration: In this phase when undertaking common processes, a use case, package and conceptual diagrams are most commonly used. Most of the system requirements are defined.

Construction: This phase is the largest phase of the process. Software is build from the requirements, the system architecture and the use cases are developed. Also in this phase many of the different kinds of diagrams are being used, e.g. a sequence, collaboration, state, activity and interaction overview.

Transition: In this phase the documentation and software itself is being delivered to a customer. The possible training of a system is also delivered and feedback is received.

As seen the Unified process is use case driven where use cases are being used to define requirements and the contest of iterations. In each iteration a certain set of scenarios or use cases are taken throughout the process: implementation, test and deployment.

The difference between the Agile and Unified process is quite remarkable. There are three artifacts in Scrum: Burn down chart, product backlog and sprint backlog. These three are the tools to follow and complete the project and make your Scrum project plan. When comparing this to the Unified process, there is a long list of artifacts and a list of documents for planning the project. In a simple way, the difference between these methods is the amount of things. Things in here means artifacts, roles, activities etc. In my opinion this all comes to that point that the Unified process requires bit more advanced and skilled developers than Agile. The difference, which is remarkable, is shown in figure 16.

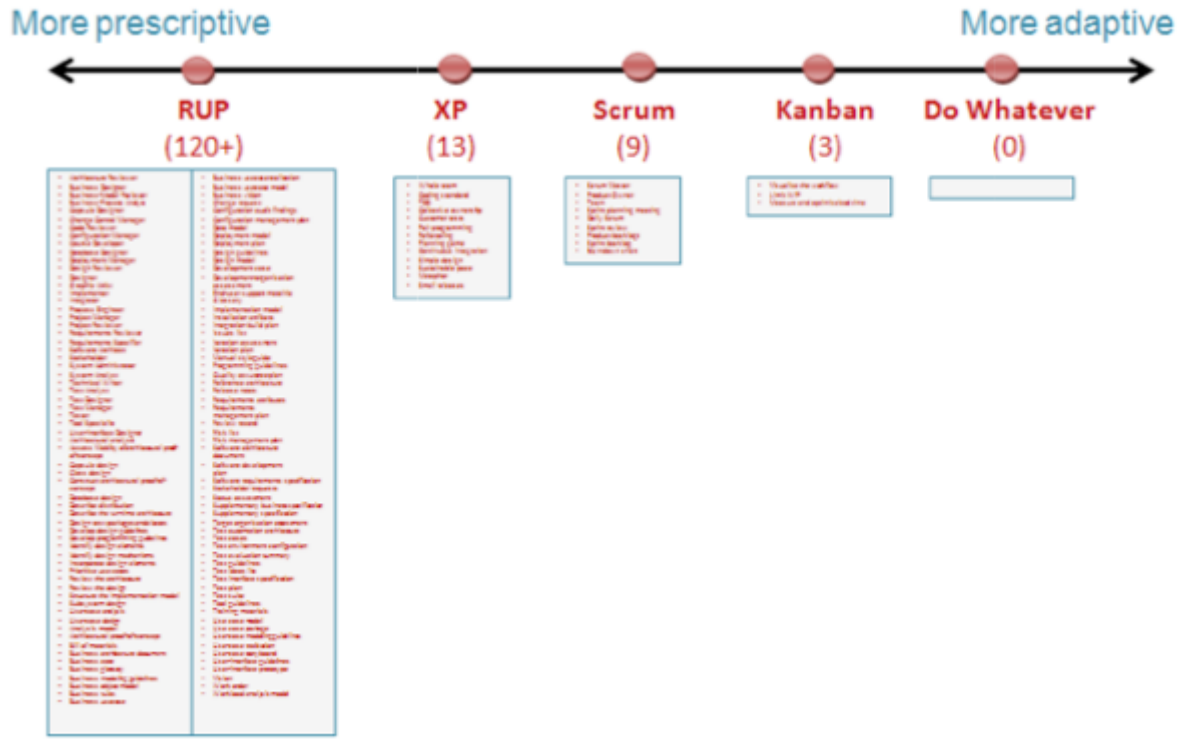

FIGURE16. Difference between Unified Process and Scrum

4.1 Agile unified process

Comparing to the Unified process to the Agile Unified process, the latter is a lighter and simpler version. Especially, what comes to artifacts and documents the Agile Unified Process differs most, saying that not all of them are needed. In general, the main idea behind the Agile Unified Process is to make Unified Process a bit more Agile to make it more streamlined, [24] like the name of process says: Following Unified Process using Agile concepts and techniques. These techniques used in the Agile Unified process are familiar from the other Agile development models like the test driven development, presented in chapter 3.4. In figure 17 the Agile Unified Process is presented.

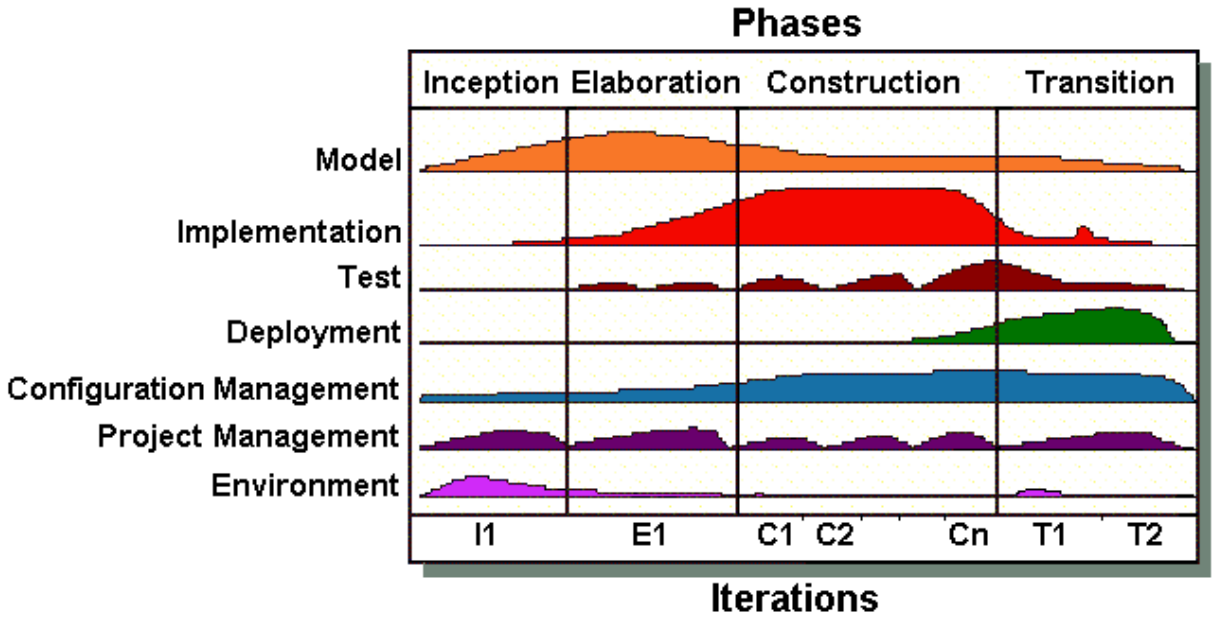

FIGURE 17. Agile Unified Process (AUP) life cycle [8].

The serial nature of the Agile Unified Process can be divided into same four phases as the Unified Process. However, in the Agile Unified Process there are

fewer disciplines. Those disciplines are performed in iterative. The disciplines are:

1 **Model**. The goal is to understand the business of the organization. A problem domain is being addressed by the project.

2 **Implementation**. The goal is to transform model(s) to the functional code. Also, to perform basic level testing and some unit tests related to the area.

3 **Test**. The goal is to make sure that the system works as specified. Including finding the defects and verifying that all the requirements are met.

4 **Deployment**. The delivery plan for the system. To execute the plan to make the system available to the end users.

5 **Configuration Management.** The goal is to manage the access to project artifacts, to track them, control and manage changes to them.

6 **Project Management.** Te goal is to direct all the activities that take place in the project. This includes risk control, directing and coordinating people, and that the system is delivered on time and inside the budget.

7 **Environment**. The goal is to have all around support for the system, so that e.g. guidance and tools are available for the team as needed.

There are also six philosophies the Agile Unified Process is based on:

1. **Your staff knows what they're doing**. Documentation and guidance are available, but nobody is forced to do that. There are also some high-level training and guidance available from time to time.

2. **Simplicity**. Everything is documented and described in a simple way. A handful of pages is enough; hundreds of them are always too much.

3. **Agility**. The Agile Unified Process follows the values and principles of the Agile Alliance.

4. **Focus on high-value activities**. The focus is on the activities that matters the most., not in every possible thing that could happen.

5. **Tool independence**. In the Agile Unified Process you can use any tools you want. It is also recommended that you use exactly those tools that suit best to your job.

6. **You'll want to tailor the AUP to meet your own needs**. The Agile Unified Process made product can easily be tailored with any common HTML editing tool. You don't need any special skills or tools to do that. [24]

5 EMBEDDED SOFTWARE DEVELOPMENT

Embedded is hard to define in a common way. Nearly every computing system is embedded excluding a desktop computer. Even if embedded devices can be almost anything there are certain common things among them. Embedded systems are always designed strictly for a certain purpose. These pre-defined features make the optimization possible for developers. The optimization can be done in many ways in both software and hardware side. In situations where hardware is customized for the certain product, it might easily lead to a situation that software is developed to hardware which may not be even close to ready. Many times when speaking of embedded systems, it can also be talked about a real-time system as well, since embedded systems are often also reactive. Embedded systems can roughly be categorized into three different categories:

Tight constrained: These are low power, low cost, fast, small and so on.

Single-functioned: These execute a single program repeatedly, over and over again.

Real-time and reactive: Continually changes because it reacts to system changes, computes and makes actions in real-time without any delay. [23]

Real-time and reactive systems mean that a system can be idle for long periods, waiting an input from the user. After receiving the input, the system reacts with a designed way. The input can be anything from a button press to giving some command in other way. Real-time systems bring more complex design issues on the tables which are needed to be solved in really early stages.

There are four different design challenges in embedded systems: unit costs and platform metrics, non recurring engineering (NRE), time to market and common design metrics. The key challenge in the embedded software development is optimizing design metrics. The common design metrics drive the embedded system design. In figure 18 common metrics in embedded software development are presented.

Unit cost: cost of manufacturing the system, not including NRE costs
NRE cost (Non-Recurring Engineering cost): Total cost of designing the system
Size: the actual physical space required by the system
Performance: the execution time of the system
Power: the amount of power consumed by the system
Flexibility: the possibility to change the functionality of the system without big NRE costs
Time-to-prototype: the time that is needed for building a working version of the system
Time-to-market: the time that is required to develop a system so that it can be released to customer
Maintainability: the ability to modify the system after it has been released
Safety: probability that system work safe way without causing any risks
Correctness: system functionality is the same than required and correct

FIGURE 18. Common design metrics [23]

The easiest way to cut down the NRE costs would be to use a powerful multi-purpose processor, but this would affect heavily the unit costs. The size of the product would most likely become bigger and the power consumption would increase. In small high volume products, products like mobile phones, the main goal is to minimize the cost.

Many embedded systems are being run by a battery. This brings limitations to power consumptions. Especially when speaking about the performance it is balancing between the power consumption and performance. Many times the power consumption is being noticed in favor of the overall performance.

Many embedded systems are designed to last many years, e.g.an air conditioner. There are also exceptions usually when speaking about small consumer devices, e.g. an MP3 player, a mobile phone. These kinds of systems are usually designed to last only a couple of years. When making embedded systems with a long lifetime it is important to notice possible issues of reliability and software updates. Nowadays software updates are not that big issue anymore. Many devices can easily be updated via an Internet connection. It is really common that when you buy a TV or a mobile phone and take it into use, the first thing you do is that you update the software.

6 AGILE TESTING

This chapter is to briefly describe what the Agile testing is all about. What kind of things are affecting the test planning or performing the tests. This chapter also includes thoughts about the Agile testing in embedded software systems; inspecting possible differences and situations that are needed to be taken into account when planning or performing tests. Different test types are listed in figure 19. [20, p.189-215]

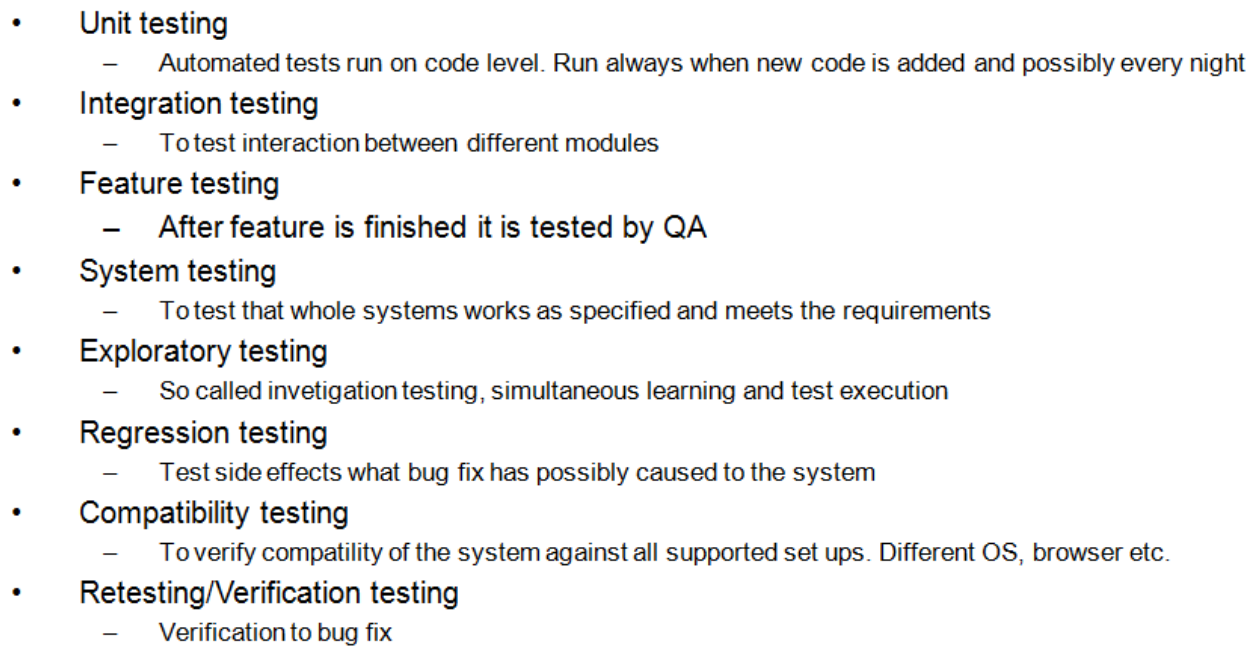

FIGURE 19. Different test types [20, p. 189-215]

The core of the Agile software development is unit testing. However, even if a strong unit testing is the key element for the successful Agile project, embedded systems bring special problems to this equation. Testing the embedded systems is a mix of testing hardware and crossing the organizational and professional limits. Having hardware in the same picture, the Agile methods work well by providing a use of multiple test strategies. This has a strong impact on increasing the quality of the embedded system. Unit tests are usually

performed by development team members. A structural unit testing is targeting to find bugs in low level operations. A well performed unit testing is a big help for an integration testing, which is to test module interfaces and how well they operate together. Automated unit tests are very effective and good because they can always be used also in a maintenance phase. To keep maintenance easy, it is important to have a good document of the unit tests.

It is important that regression testing is not forgotten. Also, developers should participate in the regression tests. If the regression testing is done properly and a test system is user-friendly it is easy also for the developers to run them, at least in a simple mode by running scripts. The developers should be encouraged to this because it helps when checking changes done against different hardware platforms before the changes are being committed to the master code.

The integration testing is usually done by the integration team, but is often also done by the development team. Testing the functional stability of the system is covered by a system testing. The system testing is performed by the test team. System integration testing is to assure how well software is able to interoperate with the other software systems specified. System integration tests are also done by the test team. A user acceptance testing is done to assure that the whole system works correctly, meets all the requirements and is formally ready to be released to the end user. This testing can be done by the end users. Often a certain group of end users is used to perform these tests.

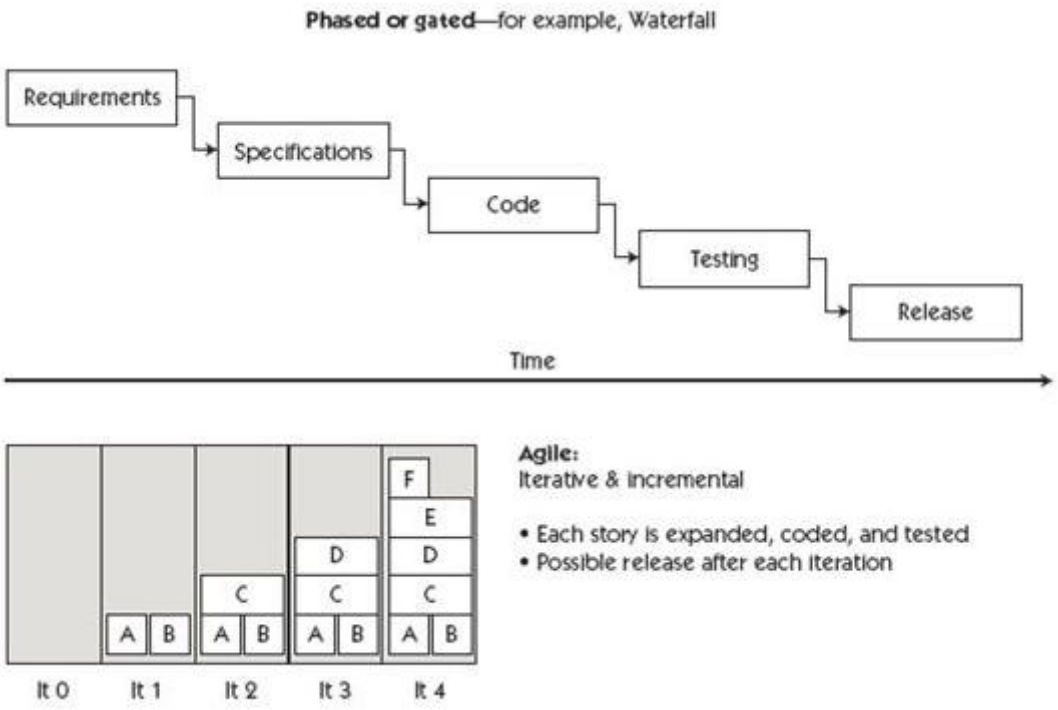

FIGURE 20. Agile vs. Traditional testing [20, p.13].

It is good to remember that programmers never go ahead testers, since a story is not finished before a program has been tested. Especially, when working in a Scrum team, it is important to really be part of everything as a tester. If you are not being invited to meetings or planning sessions something is seriously wrong. The team cannot work in that way and the team has become risk in that situation. Sometimes testers are thought as a separate part by the developer team. This is one main reason why it is recommended that a Scrum team works in the same area, close to each other. However, there are many kinds of teams out there and many different types of Agile approaches to development, some of them are mentioned in the earlier chapters; e.g. extreme programming, feature-driven development, Scrum. There are also self-titled teams that call themselves Agile, without really practicing Agile.

Lisa Crispin and Janet Gregory in their book “Agile testing – A practical guide for testers and Agile teams” have listed ten principles that they think are

important to an Agile tester. Those principles are easy to agree with. The principles are listed in figure 21. [20, p. 22]

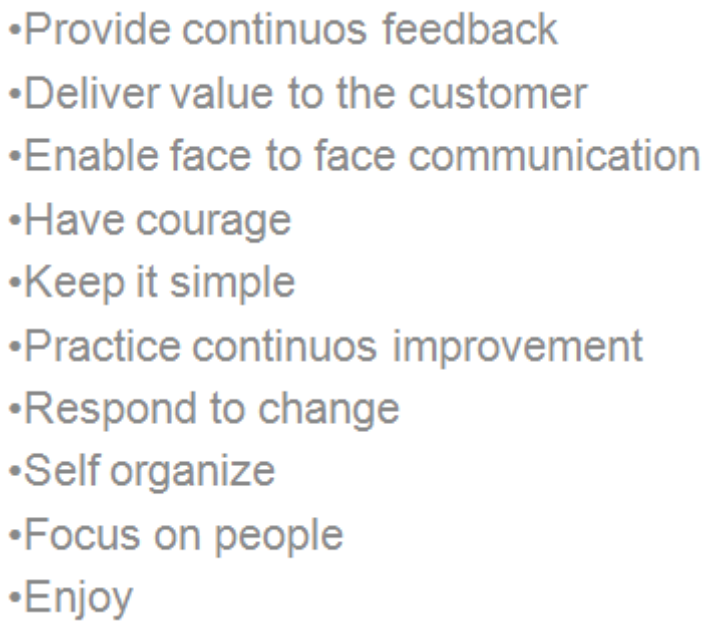

FIGURE 21. Ten important principles for Agile tester [20, p. 22]

General in Agile and especially in Scrum QA is not only the test team's responsibility. QA includes all the actions that we do to ensure a better quality and less bugs in the development process.

7 PROCESS FOR EMBEDDED SOFTWARE DEVELOPMENT

In this chapter I try to introduce a new methodology for the embedded software development based on the previous methodologies presented in this thesis. After a research, inspection and experience I will try to solve the common problems of the commonly used methodologies. First, I propose a process for the embedded software development. After that I present details about practice and retrospective. The final section considers how this process would meet the common problems and issues using the Agile development model in the embedded software development. It is good to keep in mind that these possible solutions are reflected towards my current organization.

Since hardware is sometimes available only in the very late stages of the project, I think it is really important to realize the need of a good simulator or a test environment. Even if it needs lots of commitment, it will pay off in later stages. Against the simulator and/or the test environment it is possible to develop software as you would already have the hardware available. With the simulator it is important to write lots and lots of unit tests against it. After these are done and people have become confident of this area, it is much easier to move to other Agile techniques. Still there are often cases where it is unclear if the bug is in hardware or software, but luckily some of the hardware problems can be corrected with software. This is unfortunately a quite usual step in the late phases of the project.

When designing the whole system in the very beginning, it is important to think, how it can be divided into smaller testable components efficiently. There is also another side in this: When trying to lock all the requirements in so early stages, it means that in some point you would need to say "no" to a customer.

7.1 Process

After investigating the possible processes my first thoughts were that I will end up into the customization of Scrum, extreme programming and feature driven development. The baseline came from Scrum, but it had major impacts from the extreme programming and feature driven development. Scrum is a very well working process for software projects, but in the embedded system development it needs some modifications.

Some impacts to the process I would take from the test driven development. In the test driven development it is easy to follow and track your doing. The development is efficient because it is fairly difficult to develop unnecessary things. This raises the developer's responsibility to maintain the tests. If the code changes or something is being added, the tests must be modified too. In the test driven development the maturity of the code that goes to the testers is higher than in other models. I think this cuts down the risks quite dramatically and efficiently.

At the beginning I would have wanted to use some parts from the extreme programming, since it drives heavily towards the customer satisfaction. However, I finally came to the conclusion that this would need more from developers than was possible to have. As the name says it is a very extreme way of working, thus also from that point of view I felt that it would have needed too much from the transform process. In the extreme programming it would be important to have a chance to do a pair programming. This seemed impossible in the situation that I was investigating at. I also got the picture that the extreme programming would not work very well in bigger projects, therefore it was rejected too.

In a addition to Scrum, I think that the feature driven development also offers some good things to the embedded software development. Unlike extreme programming, the feature driven development would possibly also work in bigger projects because of its scalability.

One big part of a new process is the tools used. When selecting a new process, it is very important to find and use the right tools for the process in question. This is a big investment since some tools cost quite much. I still find that it is very important to do that because without effective tools it is impossible to say if some process is effective to your team. The tools are more like a side effect of applying a new process and there are so many of them that I did not study this subject more thoroughly.

7.2 Practices

The practices, in other words the day-to-day work, are an important part which needs to be carefully considered when applying a new process. This impacts heavily on the outcome since if it is well planned, it speeds up everyday work, but in worst cases it slows things down remarkably. New practices do not necessarily mean totally new practices, a single practice itself can be old, but when combining these into one big chuck, it most likely is a new one.

In the Agile manifesto it is mentioned that "team reflects on how become more effective" [Appendix 1]. Relating to this, maybe the most important practice is to have retrospectives, in my opinion. Having retrospectives gives the team an easy way to improve their doing. Often it is easy to leave this undone since the sprint work has already been done. In the retrospective held after every sprint the team goes through the general issues how everything went, were there some problems and can they be corrected in the next sprint. Especially, when transferring to a different process model, this is really important since there are always some difficulties in the first few sprints. The team has a big responsibility and what comes to taking the possible actions from the retrospective, there is no handover. The team itself is responsible for it.

Scrum does not have everyday practices for the development. Therefore it is important to have an addition from another methodology to have support also to that side. In this subject I find the feature driven development or test driven development a really good support for Scrum. The feature driven development and test driven development also offer practices for the unit testing, which I consider as a very important thing for a successful project. At last I like the idea behind the feature driven development: The development is done by a feature. One feature in a complex system might mean that there is a development needed to be done in several components, in this way all features will be done throughout the whole pipe and can be tested completely.

7.3 Outcome and how Agile meets the requirements

Scrum itself had some really good qualities what came to running the software development project. It just needs some modifications when turning it in to the embedded software development project, e.g. unit testing and developing by feature , because embedded projects are usually quite large and very strictly structured.

For a long time it has been questionized that how well the Agile methods suit in to the embedded world. Lots of criticism has been published, but there are also some positive signals found from this area.

8 DISCUSSION

The idea and target of this thesis was to find a suitable Agile process for the embedded software development. After having been part of many Agile projects in the embedded software area and having no experienced of a fully working system yet, I was motivated to study this subject more thoroughly. This thesis started with a massive background investigation and a study of different Agile methods. There would have been even more different methods available, but I had to draw a line somewhere. Otherwise there would have been too much theory based information which is not necessarily relative to this thesis. The methods presented in this thesis were possible candidates and had some good qualities which could be used when planning the customized process.

The outcome was mostly that I ended up into estimating the process from different angles and tried to reflect them in to my experience about the problems I have met. There were interesting aspects that I noticed about the processes after studying them. For example the extreme programming seemed first really radical. From the beginning it was obvious that Scrum was the most familiar for me and I tried to use it as much as possible. A big surprise was how popular the feature driven development was and how process oriented it was. The feature driven development was at first my favorite since my previous organization was very process oriented. Another reason for the feature driven development being my favorite was that in my previous organization a big number of large projects were ongoing at the same time. Because the feature driven development promised scalability, it seemed like a really good base for the process. However, during working on this thesis my job in that firm ended and I changed to a totally different kind of organization. This affected heavily the outcome and I had to do big parts all over again since I could not reflect those processes into reality that well anymore. In that point I also had the Agile Modeling in this investigation since I wanted to bring heavily the Agile practices in to the new process. The

Agile modeling would have been rather easy to take in to use and spread across the organization. But eventually I also gave up this, because I did not see it so relative anymore.

At one point I noticed that I had several different processes since I could not find one suitable process for all teams in our organization. There were so many different kinds of teams which varied from each other heavily. It was not very likely that I would find one certain process that suits for all of them. I had to start processing this thesis from common problems I had seen during my career and I tried to solve them. I had no experience of what kind of problems developers meet in every day routines.

It would have been very beneficial to have an opportunity to do this in co-operation with my own organization. Then I could have had actual chances to see the difficulties what are met using my proposed process and I would have had a possibility to adjust it in the run. Since this was not possible, I had to settle to think these at a theoretical level. Of course, it would have also been very challenging to train people in to the new process but schedules are so tight all the time that a suitable time slot would have been almost impossible to find.

One of the biggest challenges in this thesis was a lack of information available. There is not much solid information about where the Agile methods are used in the embedded software development. From Scrum there was more than plenty of information available and also my own experience helped in this. However, information on extreme programming and especially on the feature driven development information was really limited. The problem caused by this was that it is hard to define something reliable without any proven feedback about how it is working in real life situation.

I still believe that it would be totally possible to find a suitable Agile process which suits well to the embedded software development. The point is that it would need to be modified to the organization which is planning to use it. The

embedded field is so wide and complex that by presenting one process, it is impossible to cover all possible organizations.

9 CONCLUSIONS

This thesis presented the possible options for an embedded software development process using Agile methodologies. The possibilities and the options were presented and thought only at a theoretical level. The subject grew at the same speed as the thesis was going forward. There came new points and aspects after at every corner. It was easy to realize why this subject is still quite open what comes to having an absolute final solution for all. Having a solution that would suit all projects and different companies is rather impossible. There is a possibility to customize processes is such a way that it works well in the project in question. However, this would request lots of knowledge and research before it could be taken into use. Many projects have not these kinds of resources or time in their schedule.

Because the embedded systems are more rigid when compared to some other software systems, at least in some aspects the Agile methods cannot be used universally in the same way in every place. This needs a special customization. However the benefits of using the Agile methodologies are real, especially if a company is aiming at speeding up a product's development cycle. In addition, the adaptation of the team is really important. Adopting new ways of working is always challenging. New practices and new roles ,etc. usually take lots of time to make them run in the way they should. This is important to realize when doing bigger changes to the processes.

The process of this thesis was quite long, since I have been gathering the information for years now from the projects I have been working. The theory part and all the information there were more overwhelming than I assumed, especially hardware related Agile issues remained rather low. Hardware is almost unknown area for me and therefore it was really difficult to find a solution at a theoretical level since I am lacking a work experience from that field. Whatever direction will be decided to go with the process, it is important to

maintain the retrospectives. It is important to keep the team functioning Agile and continuously improve the way of working.

At the end it is easy to summarize a few things: processes is not useful on the paper, it always needs hands in practice before it can be finally accepted, rejected or made plans for the adjustments. Secondly: the increase of success can be increased by making faster deliveries. Finally: the process cannot be made without assigning people to it.

REFERENCES

[1] Royce, W., 1970, Managing the development of large software systems,
Date of retrieval 10.05.2014
http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/Waterfall.pdf

[2] Kranz, W., An Integrated System Development Process including Hardware and Logistics based on a Standard Software Process Model
Date of retrieval 10.05.2014
http://ftp.rta.nato.int/public//PubFullText/RTO/MP/RTO-MP-102///MP-102-04.pdf

[3] Takeuchi, H., Nonaka, I., 1986. The New Product Development Game.

[4] Takeuchi, H., Nonaka, I., 1995. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation

[5] Janzen, D., Saiedian, H., Simex, L. L. C., 2005. Test-driven development concepts, taxonomy, and future direction

[6] Hartman, D., 2006. Interview: Jim Johnson of the Standish Group
Date of retrieval 12.05.2014
http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS

[7] Wells, D., 1999, Extreme programming: A gentle introduction
Date of retrieval 20.05.2014
http://www.extremeprogramming.org/

[8] Ambler, S., 2002. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process
Date of retrieval 18.05.2014
http://www.ambysoft.com/unifiedprocess/AgileUP.html

[9] Murphy, C., 2004. Adaptive Project Management Using Scrum, Methods & Tools
Date of retrieval 10.05.2014
http://www.methodsandtools.com/archive/archive.php?id=18

[10] Schwaber, K., Beedle, M., 2001. Agile Software Development with Scrum

[11] PPM Studio: Agile software development
Date of retrieval 10.05.2014
http://www.ppmstudio.com/Agile-Software-Development.aspx

[12] Nebulon Pty., Ltd. 2005. Feature Driven Development overview.
Date of retrieval 10.05.2014
http://www.nebulon.com/articles/fdd/download/fddoverview.pdf

[13] Palmer, S., Felsing, J., 2002. A Practical Guide to Feature-Driven Development.

[14] Beck, K., 2002. Test Driven Development: By Example

[15] Ambysoft inc., Introduction to Test Driven Development
Date of retrieval 22.05.2014
http://www.Agiledata.org/essays/tdd.html

[16] Standish group international, 2012. Chaos manifesto 2012
Date of retrieval 22.05.2014
http://www.versionone.com/assets/img/files/CHAOSManifesto2012.pdf

[17] Standish group international, 2011. Chaos manifesto 2011
Date of retrieval 22.05.2014
http://www.versionone.com/assets/img/files/ChaosManifest_2011.pdf

[18] Boehm, B., Turner, R., 2005. Management Challenges to Implementing Agile Processes in Traditional Development Organizations

[19] Balaji, S., Murugaiyan, M., 2012. Waterfall Vs V-model Vs Agile: A Comparative Study on SDLC.

[20] Crispin, L., Gregory, J., 2008. Agile testing: A practical guide for testers and Agile teams

[21] Johnson, J., 2002. Keynote Speech XP 2002

[22] Poppendieck, T., 2003. Agile Customer's Toolkit.
Date of retrieval 22.05.2014
http://www.rallydev.com/documents/Rally_Agile_Customers_Toolkit.pdf

[23] Grünewald, M., 2007. An introduction to embedded systems design.

[24] Grenning, J., 2007. Agile embedded software development
Date of retrieval 22.05.2014
http://www.renaissancesoftware.net/files/articles/ESC-349Paper_Grenning-v1r2.pdf

[25] Schooenderwoert, N., Morsicato, R., 2004. Taming the Embedded Tiger – Agile Test Techniques for Embedded Software

[26] International Scrum institute, 2014. Scrum burn down chart
Date of retrieval 27.05.2014
http://www.Scrum-institute.org/Burndown_Chart.php

[27] Miller, J., 2012. Live Limitless
Date of retrieval 28.05.2014
http://www.limitless365.com/2012/04/19/80-results-20-efforts/

APPENDIX 1 - AGILE MANIFESTO