David Marogy, Bsc

Development of an Approach to integrated Product Development of Hard-, Firm- and Software

Master's Thesis

to achieve the university degree of

Dipl.-Ing.

Master's degree programme: Softwareengineering and Management

submitted to

Graz University of Technology

Adviser: Univ.-Ass. Dipl.-Ing. Harald Wipfler
Auditor: Univ.-Prof. Dipl.-Ing. Dr. techn. Stefan Vorbach

Institute of General Management and Organization

Graz, April 27, 2018

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Datum / Date

Unterschrift / Signature

Thanks

I would like to express gratitude to my advisor, Dipl.-Ing Harald Wipfler for his patience and for helping me in situations in which it was not clear what I wanted. I also want to thank him for his patience concerning the discussions which took longer than usual. Additionally, I want to thank my auditor, Prof. Dipl.-Ing. Dr.techn Stefan Vorbach, who provided me with interesting papers and other sources and I want to thank him for the various discussions which lead to several ideas.

I also want to express my gratitude to the business unit segment leader of AVL Dr. Rüdiger Teichmann for giving me the opportunity of writing this master thesis and I want to thank him for several interesting ideas he promoted in the discussions we held.

Furthermore, I want to thank all my colleagues from AVL who I interviewed and who participated in interesting discussions about agile development.

I want to thank my parents for always supporting me in my decisions, my brother for his help and for various discussions. I want to thank my friends (especially Andreas) for all their support and for reading my master thesis.

Lastly, I want to thank my girlfriend who had to handle all my complaints and for encouraging me to always do my best and for always staying by my side.

Abstract

The purpose of this study is to investigate whether it is possible to use agile software development methods for developing products with hardware, software and firmware for the company Anstalt für Verbrennungskraftmaschinen List (AVL). AVL itself uses two different agile development methods: AVL's lean agile Software development process with kaizen (ALASKA) for software development and AVL's Lean and Agile Device Innovation Framework (ALADIN) for hardware and firmware development, which currently is in development. The question arouse whether it is possible to combine these two development methods or to create a new method for a team of hardware, software and firmware developers. What adaptations need to be made in order to use agile software development methods for hardware and firmware development?

Due to the new mindset and the breakthrough in the software development industry through agile development methods, AVL's combustion development department decided that it is time to use agile methods for their software and they were curious whether it would be possible to also use agile methods for their hardware and firmware development.

Therefore a literature review and a comparison of three popular agile development methods was made: Scrum, Extreme Programming (XP), Feature Driven Development (FDD). With these in mind, literature was consulted concerning agile development for hardware and firmware development. Most of the studies use Scrum as a base and adapted it to their needs. Three cases for firmware and for hardware development were found, which indicate that it is possible to use agile development methods not only for software development.

After the literature had been examined, the agile development of AVL was reviewed and compared with each other and with the results from the literature. Interviews were carried out with the hardware, software and firmware employees of AVL, to investigate what they think about agile development, how it can be adopted to their needs and what challenges may occur.

The results of this study show that it is possible to use agile software development methods for hardware and firmware development. The Scrum framework can be used and can be adapted to the needs of hardware and firmware development. Due to the constraints of hardware, adaptations must be done in the planning stage to reduce delays in the product delivery which arise from shipping and production delays. The development methods of AVL can be used and combined together to one framework with one development team containing hardware, software and firmware developers. Another possibility would be to use only a small variant of this development method.

Changing the mindset of the employees is a vitally necessary factor for the success of the agile development method, which needs time and a managerial lead. Nevertheless, adaptations must be done to ensure that this method creates value.

Contents

Figures

Tables

Abbreviations

AVL	Anstalt für Verbrennungskraftmaschinen List
ITS	Instrumentation and Test Systems
XP	Extreme Programming
FDD	Feature Driven Development
OOPSLA	Object-Oriented Programming, Systems, Languages and Applications
MUSCOW	Must have, should have, could have and‘won’t have this time
ETVX	Entry criteria, Task, Verification, Exit criteria
UML	Unified Modeling Language
EPROM	Erasable Programmable Read-Only Memory
ROM	Read-Only Memory
VCC	Volvo Car Cooperation
OEM	Original Equipment Manufacturer
PAL	Processor Abstraction Layer
CAD	Computer-Aided Design
NASA	National Aeronautics and Space Administration
SAAB	Svenska Aeroplan Aktiebolaget
EDS	Electronic Data Systems
GRB	Garðabær
PIP	Product Innovation Plan

ALASKA	AVL’s Lean Agile Software Development Process with Kaizen
ALADIN	AVL’s Lean and Agile Device Innovation Framework
WSJF	Weighted Shortest Job First
MI	Measurement and Instrumentation Indicating
MIS	Measurement and Instrumentation Support
MIE	Measurement and Instrumentation Engineering
PLC	Product Life Cycle
SAFE	Scaled Agile Framework

1. Introduction

In cooperation with AVL agile software development methods are reviewed. It is tested whether these development methods can also be used for not only software products but also for products with a combination of software, firmware and hardware.

The software industry has changed over the past years. More and more companies are switching from the classical development method to agile development methods. However, before agile software development, it was common to develop software by using a sequential approach such as the waterfall model. (Holtsnider et al., 2010)

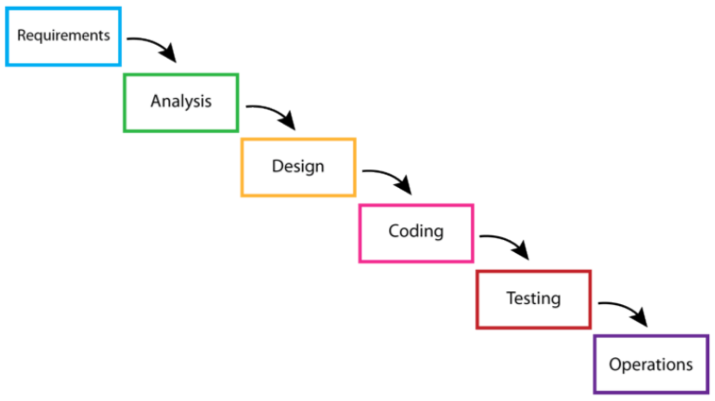


Figure 1.: Waterfall model (Reynisdottir, 2013)

The waterfall model contains different phases, such as planning, implementing and testing. As the name already implies, this is a way of developing software by finishing phase after phase and gates after gates until the product is finished (figure 1). These phases and their gates depend on the cost of change(Holtsnider et al., 2010).

Problems in one phase or gate may lead to a missed target date, or a product which does not meet the needs of the costumer. Therefore a new way of developing software was needed; the agile software developing method.

In February 2011, seventeen recognized software development “gurus” met to find a common ground for developing good software in a ski resort in the Wasatch Mountains of Utah. In this month the manifesto for agile development was born (Highsmith, 2002). The participants agreed on the declaration of the manifesto which states: *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- ***Individuals and interactions** over processes and tools*
- ***Working software** over comprehensive documentation*
- ***Customer collaboration** over contract negotiation*
- ***Responding to change** over following a plan.*

That is, while there is a value in the items on the right, we value the items on the left more".(Alliance, 2013) Additionally to these values, 12 principles were declared to state what it means to be agile. These principles are listed in Alliance (2013).

However, there are other areas of development. Companies who develop products with hardware, software and firmware are currently using classical development methods. These products have high uncertainty.

As an employee of AVL, I heard that in my department the demands for using agile development methods for product development with hardware, software and firmware is highly increasing. The question aroused if it is possible to use such methods for our purpose. A study was needed. As i was seeking for a master thesis, I considered this an opportunity to write about something practical and I asked my business unit segment leader Dr. Rüdiger Teichmann if I could elaborate on that.

1.1. Purpose

The purpose of this thesis is to investigate whether it is possible to use agile software development methods also for products which are a combination of software, firmware and hardware for the segment Measurement and Instrumentation Indicating (MI) and its departments Measurement and Instrumentation Engineering (MIE) and Measurement and Instrumentation support (MIS) of AVL. Therefore some of the popular agile development methods are analyzed and compared. Based on best practices it will be tested, what adaptations need be done in order to use them in a more firmware/hardware based environment.

For each part it is investigated which agile development methods are used, what adaptations are made for using them in an different environment and also some best practices are reviewed.

Additionally, interviews with the employees of the AVL are conducted and transcribed. These interviews help to understand what the employees are thinking about the change from a rather waterfall orientated method to an agile development method, what challenges exist and what advantages result from using them.

Based on these results, the challenges for adaptations are analyzed and some proposals are made to overcome these challenges.

1.2. Problem Formulation

In this section, the problem is formulated and a small introduction of AVL is done. Based on the problem definition, research questions are formulated. These research questions will be answered by this study.

1.2.1. Introduction of AVL

AVL is the leading company for development, simulation and testing of powertrains for cars, trucks and large engines. The segment MI for which this thesis is made, provides combustion analyzing platforms with hardware and software components. The segment is splitted into the MIE department for developing hardware and software for combustion analyzing platforms and the MIS department for service and support for customer who bought the combustion analyzing platforms. For its latest product X-ION, software, hardware and firmware employees collaborated in order to create it. Recently, it has been decided to create some new modules for this product.

1.2.2. Problem Definition

Before this study was conducted, the software, firmware and hardware teams collaborated based on an own product innovation plan (PIP). This PIP is a waterfall based method, contains many gates and constraints on how to plan and what kind of steps are necessary to get the product on the market. Because of this and due to other reasons, managers at AVL decided to use agile development methods instead of classical development methods. There are two different agile development methods at AVL: ALASKA for software products and ALADIN for firmware and hardware products. Since the product of this department contains software, hardware and firmware a new way must be found to develop it more efficiently (by using these two agile development methods from AVL). Challenges occur for products with a combination of hardware, software and firmware. Hardware has some physical constraints, such as ordering parts, delivery times, and creating physical objects. A way must be found to test firmware and software if a hardware prototype is not available. Therefore these two development methods must be analyzed to see whether it is possible to combine them to just use one or to not use any of the products.

1.2.3. Research Questions

This thesis will answer research questions which occurred during a discussion with the business segment leader Dr. Rüdiger Teichmann:

- Is it possible to use agile software development methods for products with hardware,software and firmware development?
- Does the agile software development method need some adaptations in order to be used in hardware and firmware development, and what has to be changed?

The aim of the research questions is to investigate first if it is theoretically and practically possible to use agile software development methods for products with hardware, software and firmware. Furthermore, it is investigated whether adaptations to the agile framework are needed in order to use them for developing products with hardware, software and firmware at the AVL.

1.2.4. Research Approach

For answering the research questions, the approach was to have a literature review at the beginning. After that the agile development processes from the AVL were analyzed and expert interviews were hold (figure 2).

In the literature review, agile software development methods such as Scrum, XP and FDD were investigated and compared in order to gain an understanding of how agile development works for different methods. Furthermore, a literature review was carried out for agile methods in a non-software environment. For each field (hardware, firmware) practical examples were examined for adaptation of agile software development methods.

After finishing the literature review, the processes (ALASKA and ALADIN) were analyzed and compared with each other and also with the literature. Based on these an understanding of how the agile development of AVL works is established. This comparison also shows the differences there are in the literature and what kind of proposals from the literature could help to adapt the AVL processes to our needs.

An empirical study was conducted with the employees of AVL. The employees are hardware, software and firmware developers, project leaders, testers and process managers. The purpose was to evaluate the opinion of the employees in general for agile development methods and for using them for developing product with hardware, software and firmware; for challenges which could arise from implementing them and for solutions for solving these challenges.

After that a discussion was held with all the results from the previous steps, challenges and

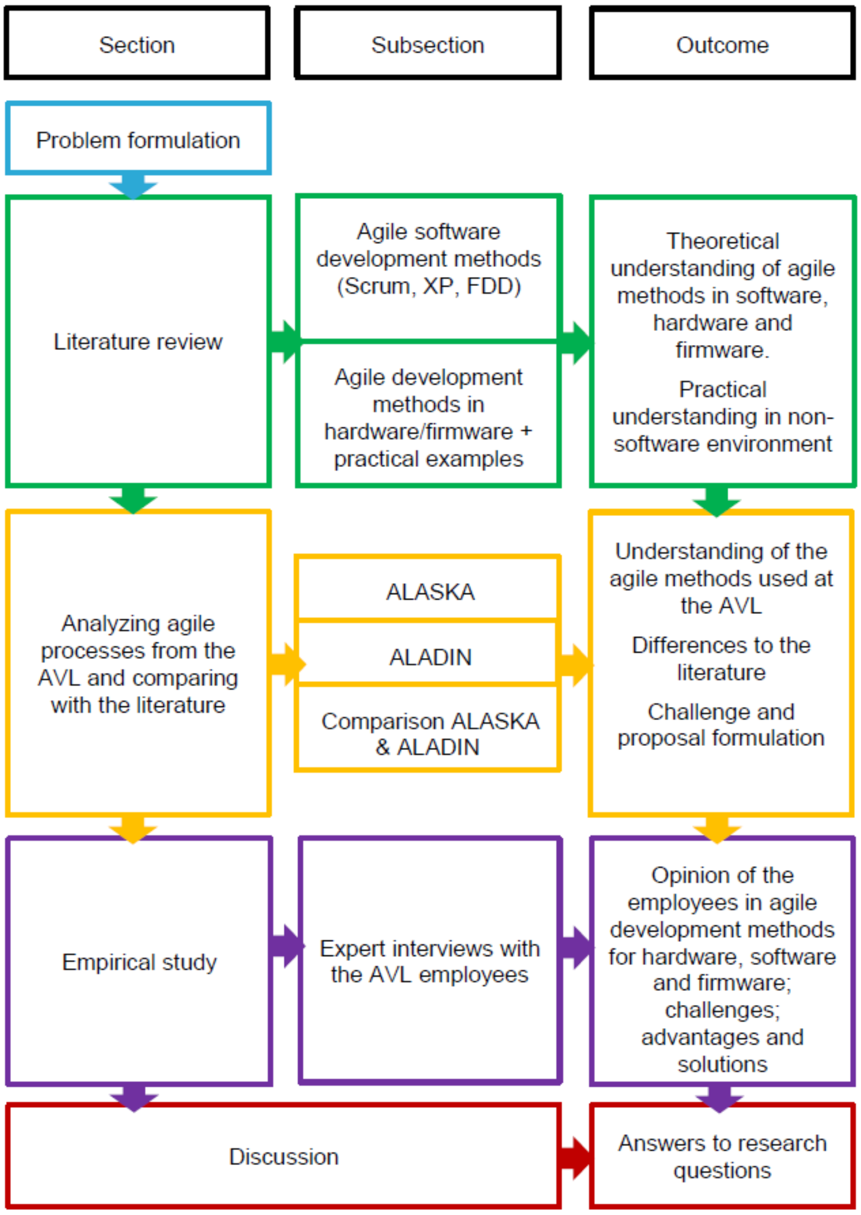

Section
Subsection
Outcome
Problem formulation
Literature review
Agile software development methods (Scrum, XP, FDD)
Agile development methods in hardware/firmware + practical examples
Theoretical understanding of agile methods in software, hardware and firmware.
Practical understanding in non-software environment
Analyzing agile processes from the AVL and comparing with the literature
ALASKA
ALADIN
Comparison ALASKA & ALADIN
Understanding of the agile methods used at the AVL
Differences to the literature
Challenge and proposal formulation
Empirical study
Expert interviews with the AVL employees
Opinion of the employees in agile development methods for hardware, software and firmware; challenges; advantages and solutions
Discussion
Answers to research questions

Figure 2.: Approach for answering the research questions (personal design)

proposals were stated and the research questions were answered with these results.

1.2.5. Outline of this Thesis

This thesis outline is split into seven chapters (figure 3):

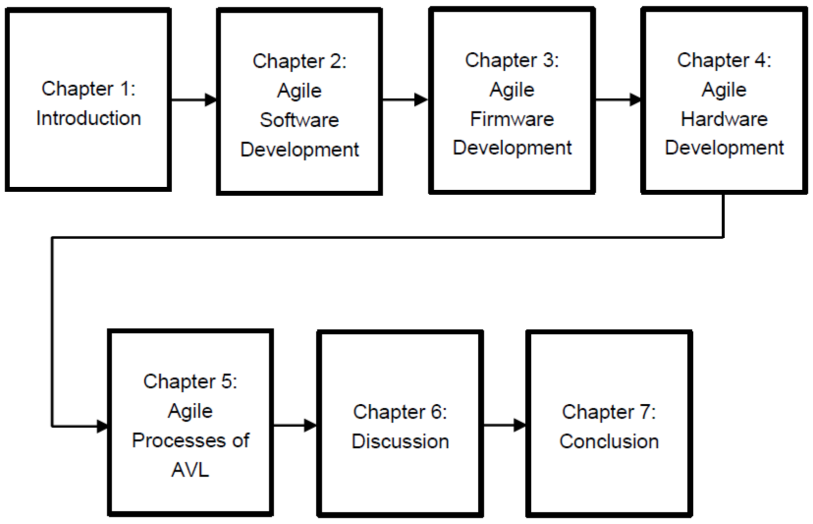

Chapter 1: Introduction
Chapter 2: Agile Software Development
Chapter 3: Agile Firmware Development
Chapter 4: Agile Hardware Development
Chapter 5: Agile Processes of AVL
Chapter 6: Discussion
Chapter 7: Conclusion

Figure 3.: Chapter Overview (personal design)

In chapter 2 an overview of the chosen agile software development methods is provided. In order to gain information, a literature review was performed to about agile software development methods. A comparison at the end of the chapter provides a better overview of the differences between the software development methods.

In chapter 3 a literature review was carried out in order to investigate what firmware development means and how agile software development methods can be adapted to the needs of firmware development. Three practical examples are explained in which agile software development methods were adapted and tested to have agile firmware development. At the end of the chapter a summary of all the information and results of chapter 3 is provided.

In chapter 4 a literature review was carried out to gain information about what hardware development means and how agile software methods can be adapted to the needs of hardware development. Three practical examples are explained in which agile software development methods were adopted and tested to have agile hardware development. At the end of this chapter a summary of all the information and results of the chapter 4 is provided.

In chapter 5 the two processes from the AVL, ALASKA and ALADIN are analyzed and compared with each other.

In chapter 6 the two processes from the AVL are compared with Scrum from the literature and an overview of the differences is provided. Furthermore in this chapter the empirical study is presented and from this study and from the results of the previous chapters, challenges for implementing agile software development methods for products with hardware, software and firmware development are discussed. Additionally, proposals for the AVL are provided with three different approaches in using the AVL frameworks; combined, separated or downgraded. Lastly, the research questions are answered in this chapter.

In chapter 7 the conclusion is presented along with suggestions for adaptations to the AVL frameworks.

Part I.

Theoretical Part

2. Agile Software Development

The following chapter provides a description of agile software development. It will provide an overview of the used methods in agile software development such as:

- Scrum
- Extreme Programming
- Feature Driven Development

2.1. Scrum

Scrum is an agile framework, for managing complex projects. The method was presented by Schwaber and Sutherland at the Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) 1995, a conference for object-oriented software development (Schuh, 2005). It is used for projects which need to be finished quickly or for projects where the requirements are not yet entirely discovered. Instead of developing in a phase, Scrum uses an incremental design and develops in iteration. Scrum contains different roles and events. These roles interact with each other in meetings and events. Projects start with planning the customers' requirements. These requirements will be set into the product backlog. The development team will then start with prioritizing them, which is important as they will decide which functionality should be in the first iteration or "Scrum sprint" as it is called. One sprint may last up to thirty days at the maximum. After each sprint there should be a working prototype of the software containing all the functionalities (figure 4). Scrum does not provide any software development methods. Instead, the development team-member can choose on their own how to develop the functionalities. There is a standoff meeting every day, in which each of the developer can provide an overview of his or her progress. Additionally, the developer may explains what he or she is planning to finish. (Sandhaus et al., 2014)

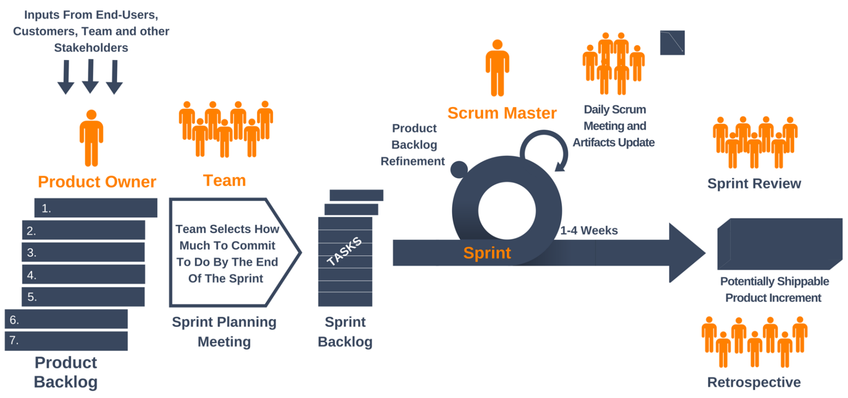

Inputs From End-Users, Customers, Team and other Stakeholders
Product Owner
Team
Scrum Master
Daily Scrum Meeting and Artifacts Update
Product Backlog Refinement
Sprint Review
1.
2.
3.
4.
5.
6.
7.
Product Backlog
Team Selects How Much To Commit To Do By The End Of The Sprint
Sprint Planning Meeting
TASKS
Sprint Backlog
Sprint
1-4 Weeks
Potentially Shippable Product Increment
Retrospective

Figure 4.: Scrum process (Schindler, 2010)

2.1.1. Scrum Team

In Scrum there are three roles:

- Product Owner
- Scrum Master
- Development Team

There is no other chef or project leader, who tells the team what to do.

2.1.1.1. The Product Owner

According to Schwaber & Sutherland (2017) the product owner is responsible for maximizing the value of the product and the work of the development team. He is responsible for managing the product backlog. The product owner's main duties are:

- Creating product backlog items.
- Ordering items to achieve goals and missions in the best way.
- Optimizing the value of the work the development team performs.
- Ensuring that the product backlog is visible, transparent and clear to all and it should show what steps need to be followed next.
- Ensuring that the development team understands the product backlog items.

(Schwaber & Sutherland, 2017)

The product owner has to decide which functionality is necessary but he or she does not enforce the rules, as this is the job of the Scrum master. Only one person can be Product Owner. This person will represent the whole project stakeholders. According to Meyer (2014) the project benefits from separating the defining of the project objectives from the evaluation.

2.1.1.2. Development Team

According to Wirdemann (2011) the team develops the software and is responsible for finishing the requirements of the sprints. Thus, it is crucial that the team is able to manage itself. This differs greatly how projects, are usually manages, as the project manager usually tells the team what to do. Here, the team organizes itself in a way that it can decide when to start developing and how to develop. Additionally, there are no hierarchies within the team. Each member can lead the team if this is necessary. A team does not merely consist of developers but rather constitutes a bunch of different skills gathered together. These skills are necessary for creating the product. There are no titles, hierarchies or leaders in a team. On the contrary, anybody can lead it if his or her expertise is needed. Schwaber & Sutherland (2017) describe these characteristics as self-organizing and cross functional. Wirdemann (2011) also states that a team should consist of at least one to two senior developers. According to Schwaber & Sutherland (2017) the size of the development team should be small enough to remain flexible but large enough complete significant work during a sprint. Having less than three team members could lead to a loss of communication and to a loss of productivity due to skill constraints. Having more than nine members would make the coordination of the team more difficult. Having five to nine members without the product owner and the Scrum master would be a good number. Wirdemann (2011) suggest that if it is necessary to have more than nine members, the team should be split up into two teams.

2.1.1.3. Scrum Master

The Scrum master is responsible for ensuring that Scrum is understood. He or she makes the team adhere to the Scrum theory, all the practices and the rules.

Scrum masters are often misjudged as project- or team leaders. However the Scrum master does not command. He or she rather makes decisions if necessary.Wirdemann (2011) describes the behavior of a Scrum master as a “Servant Leader”. Schwaber & Sutherland (2017) explains the main task of Scrum masters further. They have different responsibilities towards the product owner, the development team and the organization. The product owner has the responsibility towards the Scrum master to help him or her managing the product backlog in terms of finding good techniques and understanding how to maximize the value. The Scrum master also ensures that Scrum and agility is enforced. He or she has to help the developing team to avoid or remove any impediments to the progress of the team. Furthermore, he or she has to help the team with self- organization and cross-functionality. It is important that the Scrum master also ensures that the team is able to use Scrum in an environment in which agility is not adopted or understood. (Schwaber & Sutherland, 2017)

2.1.2. Scrum Events

The meetings and the development process from the Scrum process (figure 4) are called Scrum events. Everything important such as planning, developing and testing happens in these events.

Sprint Plan meeting: Before starting a sprint, it has to be planned which requirements should be met during the sprint. In collaboration with the product owner, the development team and the Scrum master will decide what needs to be done during the sprint. According to Schwaber & Sutherland (2017) the sprint planning should not exceed more than eight hours for a one month sprint. The meeting consists of two different parts:

- Goals of the upcoming sprint
- Designing how to do the work

In the first part of the meeting the goals for the upcoming sprint are discussed. The product owner will present the use case stories according to the priority of the backlog. The development team will then analyze and decide which stories should be considered for the upcoming sprint. They have to consider how many of the stories can be implemented in the given time. It is their choice. The selection of the items cannot be changed anymore, only the backlog may differ. (Wirdemann, 2011)

After finishing the setting of the goal for the upcoming spring, the development team will create the software design and they will divide the stories into smaller tasks which can be accomplished in one day or even less. It is also stated that if the development team has too much or too little work, the items can be changed with the help of the product owner. (Wirdemann, 2011)

At the end of the planning the development team should know how to work in order to finish the sprint and to achieve the goal. According to Schwaber & Sutherland (2017) they have to explain it to the product owner and to the Scrum master.

Sprint: In Scrum, the implementation work is done during the sprint. The team members can decide on their own which tasks they would like to work on. Additionally, they are self- organized. A sprint is maximal thirty days long. These days consist of sprint planning, daily scrum, developing, sprint review and sprint retrospective. (Schwaber & Sutherland, 2017).

Schwaber & Sutherland (2017) list some points during the sprint:

- No changes are made that would endanger the sprint goal.
- The quality of the goals does not decrease.
- The scope can be negotiated again if more information is available.

Meyer (2014) also states that it is highly important that the backlog does not grow during a sprint, even if a manager finds that a function is extremely important. This is realized by using the short duration of 30 days. If a requirements is important it will be set for the next sprint. There is also the possibility for urgent cases to terminate the sprint. This decision can be made only by the product owner.

Daily Scrum: Every day the team will meet for a standup meeting, the daily Scrum. Schwaber & Sutherland (2017) argue that this meeting should be held while standing in order to keep it short. They suggest that the meeting should be maximal fifteen minutes long and that it should be held at the same place and time every day. Each team member should then answer few questions:

- What have I accomplished since the last daily Scrum?
- What do I plan to do until the next daily Scrum?
- What hindrances do I see?

(Schwaber & Sutherland, 2017)

This meeting is important as it ensures that every member knows what the other team members are doing. This helps the team members who have a problem as they can solve these problems with the help of other team member, but this will not be done in this meeting. Additionally, this enables the Scrum master to find hindrances and to evaluate whether team members work on other things than on the tasks for the sprint. This again helps team members to learn to only develop things which have been arranged before for this sprint. (Schwaber & Sutherland, 2017)

Sprint Review: The sprint review is the last point of a sprint. In a sprint review, the whole team describes what has been done and what the result of the sprint is (Schwaber & Sutherland, 2017). According to Wirdemann (2011) the Scrum master should reserve a room with a projector for a review. He will have to send an email to each member and will then state the goal of the sprint and the user stories. Every participating active member will have to invest some time for preparation. One of the team members will then present the stories and the product owner will have some time to think about the user stories for the next sprint. Schwaber & Sutherland (2017) describe the review as a four hour timed meeting for a one month sprint. Shorter sprints will entail smaller meetings. The Scrum master will be responsible for creating this meeting and also for ensuring that everybody attends it.

Schwaber & Sutherland (2017) list elements a sprint review should contain:

- *“ Attendees include the Scrum team and key stakeholders invited by the product owner.*
- *The Product Owner explains what product backlog items has been “Done” and what has not been “Done”.*
- *The development team discusses what went well during the sprint, what problems it had to handle, and how those problems were solved.*
- *The development team demonstrates the work that has been “Done” and answers questions about the increment.*
- *The product owner discusses the Product Backlog. He or she projects likely completion dates based on progress to date.*
- *The entire group elaborates on the next step, to ensure that the sprint review provides valuable input to subsequent Sprint Planning.*
- *Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next.*
- *Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product. ”*

(Schwaber & Sutherland, 2017, p.13)

According to Meyer (2014) the purpose of this review is to reflect on the results and beyond, but not on the process.

Retrospective: The purpose of the retrospective is to review what went well and what went wrong in this sprint, but also to find some aspects to improve for the next sprint (Meyer, 2014).

Schwaber & Sutherland (2017) describe the purpose of a review:

- *“ Inspect how the last Sprint went with regard to people, relationships, process, and tools.*
- *Identify and order the major items that went well and potential improvements.*
- *Create a plan for implementing improvements to the way the Scrum team does its work. ”*

(Schwaber & Sutherland, 2017, p.14)

This should take approximately three hours with a one month sprint, shorter sprints should also have shorter reviews. The Scrum master, the development team and the product owner should attend the retrospective meeting, as this review includes a discussion on how to improve certain aspects. The improvements found in this meeting will then be implemented in the next sprints.

2.1.3. Scrum Artifacts

In Scrum, artifacts are described as the work or the value which provides transparency and opportunities for inspection and adaptation. They provide a maximum of transparency for key information in order to ensure that everyone understands the artifact in the same way. (Schwaber & Sutherland, 2017)

User Stories: User stories describe the functionality of a product, seen by the user and stated by the product owner. Meyer (2014) claims that a standard emerged which describes how to create a user story. The user story consists of:

- A category of user
- A goal
- A benefit

Story Points: Story Points are a measurement for estimating how much work is needed for solving the user story. The unit may vary; it can take a day of work or several hours. Meyer (2014) suggests using the difficulty of the easiest user story. Meyer (2014) describes three important properties :

- They do not have an absolute time value, therefore they are relative.
- Only fully implemented user stories can count as already achieved points.
- Only artifacts that will be delivered can count as progress.

Velocity: True velocity is the sum of story points of all finished sprints and implemented user stories. Additionally, there is a relative velocity which is the sum of the story points which can be achieved in a sprint. The relative velocity provides a good perspective of how many user stories can be set for the next sprint and for the product owner for creating a release plan. The relative velocity can be calculated as the true velocity from the last sprint or the median of all sprints. (Wirdemann, 2011)

Story Card: There are many tools for creating such story cards (from analog to digital versions). The Scrum team mostly uses the paper version due to visualization and handling reasons. A user story describing the task is written on each story card.

Story Board: A story board is a great way to raise the team's awareness of what has already been done and what else needs to be done. Meyer (2014) claims that using a story board helps to accomplish some goals in agile development:

- Picking a task and assigning it to a developer
- Keeping Track of velocity(story points implemented per iteration)
- Increasing moral of the Team
- Reducing unwanted implementation

It is common to use sticky papers on a white board. The white board is separated into different columns. Each column represents the state of the current task. According to Meyer (2014), there are the following states: to-do, in-progress, under-test and done. It is only effective if the team is at the same place, members at different locations will have to find a different way of resenting their states. There are many digital tools for this.

Product Backlog: The product backlog is a collection of prioritized user stories. The product owner is the only person who can edit a product backlog, but he or she is also responsible for making it available to every Scrum member or stakeholder. At the beginning the product owner will have to create the first version of the product backlog based on some initial talks or workshops. Wirdemann (2011) describes how a user story can be prioritized. One method is MUSCOW; “must have”, “should have”, “could have” and “won’t have this time“. It would be much more complicated to use the value, costs, customer satisfaction, the risk and the dependency on other aspects.

According to Wirdemann (2011), the user stories can then be divided into: data, effort, research interest, quality, user role, acceptance criteria and technical requirements.

The backlog is often used and split up into three parts: to implement, in progress and implemented. Meyer (2014) also states that some teams add a fourth category: to be verified.

Burndown: Burndown is a visual representation of the team’s progress. It displays the trend of remaining work over time. The time is measured as working days and the remaining work is measured in story points. (Meyer, 2014)

2.2. Feature Driven Development (FDD)

FDD is a process designed to deliver frequent, tangible, working results repeatedly (Palmer & Felsing, 2002). FDD is an agile software development methodology. It was first used at the Singapore project and developed by Jeff De Luca while working with Peter Coad on the project. According to Palmer & Felsing (2002) FDD is a system which uses simple, easy-to-understand and easy-to implement methods, problem-solving techniques and reporting guidelines. According to Highsmith (2002) the developer of this methodology describes FDD as ”client-centric, architecture-centric and pragmatic“, while Palmer & Felsing (2002) characterize it as ”having just enough process to ensure scalability and repeatability and encourage creativity and innovation all along the way”.

FDD is an iterative development process which delivers working results frequently, while having a good quality with each step at the same time. It provides status and progress information and also gives the developer not too many interruptions. (Palmer & Felsing, 2002)

2.2.1. Key Project Roles

One aspect of FDD is that if we want to use it, we also have to consider the roles of the project team. In reality, teams are mixed up with different weaknesses and strengths. FDD describes how to combine each role in order to achieve highly utilized strength and support when needed. (Palmer & Felsing, 2002)

The six key roles for a project according to Palmer & Felsing (2002) are:

- The Project Manager
- The Chief Architect
- The Development Team
- The Chief Programmer
- The Class Owner
- The Domain Expert

There are also some supporting roles, such as:

- The Domain Manager
- The Release Manager
- The (program) Language Guru
- The Build Engineer
- The Toolsmith
- The System Administrator

and some additional roles:

- Testers
- Developers
- Technical Writers

2.2.2. FDD Best Practices

FDD provides some best practices; each of the practices reinforces the other. Not all of the practices must be used but it would not be a FDD process otherwise. FDD requires using all of them even if it seems unnecessary. (Palmer & Felsing, 2002)

According to Palmer & Felsing (2002) the Practices are:

- Domain Object Modeling

- Developing by Feature
- Individual Class (Code) Ownership
- Feature Teams
- Inspection
- Regular Builds
- Configuration Management
- Reporting/Visibility of Results

Domain Object Modeling: Domain object modeling consists of a class diagram and sequence diagrams. Combining these two enables to see how each object is interacting with other objects. The advantage here is that for every developer who is working on a project it becomes clear how the solution for a problem should look like. There are no misunderstandings in terms of what should be implemented. Problems are firstly broken down into classes or objects, which are simpler to solve. For adding new functions, the domain object modeling provides a framework that helps to add functions "feature by feature". This avoids wasting time for refactoring classes. (Palmer & Felsing, 2002)

Developing by Feature: *"The term feature in FDD is very specific. A feature is a small, client valued function expressed in the form: <action> <result> <object> with the appropriate preposition between the action, result, and object.* (Palmer & Felsing, 2002, p.41)

Requirements are stated according to Palmer & Felsing (2002) in a customer friendly form and are then broken down into so-called features. Each feature should be small enough to be implemented within two weeks. Every feature which would take longer, will be broken down into smaller features. Palmer & Felsing (2002) provide examples of features such as calculating the total of a sale.

Individual Class (Code) Ownership: Class(code) Ownership means that only one person is responsible for the content of a whole class. Object oriented programming language uses classes to provide encapsulation. Code ownership here becomes class ownership, where each class is assigned to a specific programmer. (Schuh, 2005)

Palmer & Felsing (2002) state that due to class ownership only one person has the duty of ensuring that a class does not lose its purpose through modification and that this person is an expert in his class, able to explain how it works. He or she is able to implement new features into the respective class much faster than other people, which are not responsible for the class. Additionally, he or she can be proud of doing something good.

Feature Teams: As with the individual class ownership, also with feature teams a better developer is chosen to be responsible for one or more features. Each feature consists of classes with a class owner. The feature owner or team leader has the responsibility to deliver the finished feature. To do that, he or she will have to form a team in order to ensure that every class owner who needs to code a feature is in his or her team. (Palmer & Felsing, 2002)

Palmer & Felsing (2002) list four different kind of options for different ways of creating such teams.

- Going through all features, and picking all class owners involved in it, then separate it into exclusive sets
- Ask the class owner to make changes in their code according to our needs.
- Instead of class ownership, we use collective ownership.
- Change team memberships according to the current feature to be implemented.
- Creating a team for each feature and dispending it after the feature is implemented.

Each of the options has its advantages and disadvantages, but only the last seems more realistic. Palmer & Felsing (2002) also state that due to the small features the team should be also small containing only three to six people. The team also has the entire ownership of the code, which means there is no other external class owner for this feature. The class owner can be part of more than just one team; there is no restriction to this.

Inspection: Inspections are one of many ways to improve the quality and reduce the error rate of a code. In a feature team, different developers will have a look at the code of another developer and check for errors. If possible, they will find simpler way of solving the problem. Depending on how important the inspected code is, different team members are involved. For crucial parts, the chief programmers and developers will be there to verify the code and the design. All other parts are inspected with only the involved feature team members. (Palmer & Felsing, 2002)

Regular Builds: Regular Builds are crucial for verifying whether the system is working correctly or not. All the components, which are completed, are built and tested together. Depending on how often the regular builds are, errors that occur by adding new features can be detected. In most cases, the regular builds can be automated with some software tools such as Jenkins. (Palmer & Felsing, 2002)

Configuration Management: Version control systems are an important tool for keeping track of the history of changes made to software and other artifacts. In FDD a CM is only used for already finished codes of features, in order to keep track of their history of changes. Most developers only put a code under version control. Each artifact that is maintained during the development process should be put under version control, even contracts and requirement documentations. (Palmer & Felsing, 2002)

Reporting/Visibility of results: Palmer & Felsing (2002) state that managers and other project leaders need information on the current progress status. Developers often tend to implement without reporting what has been done and what the current status is. In FDD there are tools and frameworks which help to ensure a high visibility of status information of the project. That progress information should be delivered in a frequent and accurate way in order to help the leader to steer the project into the right direction.

2.2.3. FDD Process Model

The FDD consists of different processes (figure 5). Highsmith (2002) claims that one needs a system for building systems in order to scale to larger projects. Instead of using big complicated processes, small and simple processes with logical steps will be more efficient. Additionally, the focus should never be on the process itself. Highsmith (2002) states that good processes move to the background, in order to enable team members to focus on results while using short, iterative, feature driven life cycles.

Highsmith (2002) states (based on Jeff De Lucas process article) that there are only five processes at high level:

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

These processes are performed one after another (figure 5). Each process is performed once, except the processes design by feature and build by feature, they are performed as iterations with different events (figure 6).

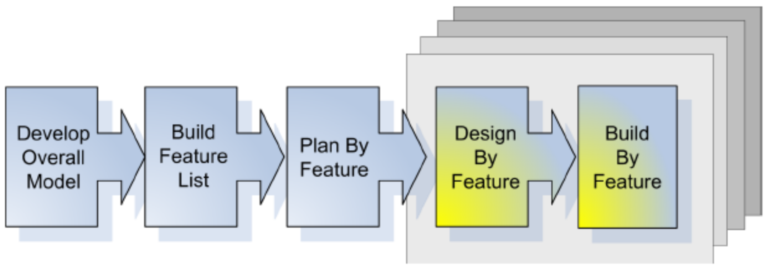


Figure 5.: Feature driven development process (Schindler, 2010)

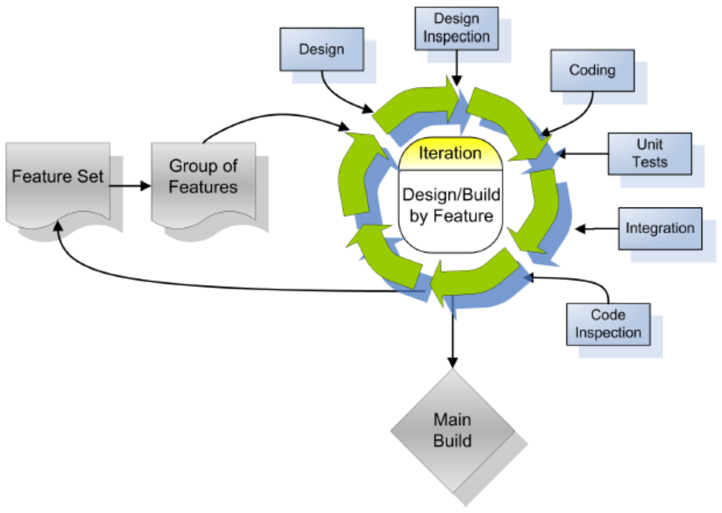


Figure 6.: Feature driven development design and build by feature process (Schindler, 2010)

2.2.3.1. Develop on Overall Model

In this phase, the focus lies on the high level, rather than on details. A skeleton of the projects is developed in this phase by creating shape models for subject areas and then combining these models into an overall model during an "integration" modell meeting. The team for larger projects consists of modelers and domain experts, led by a chief architect. (Highsmith, 2002) FDD provides a planning guideline for how long one process should last. It would take about two weeks for developing an overall model for a six month-project. Each process is based on an ETVX (entry criteria, task, verification, exit criteria) pattern. This pattern is taken from the article of Jon de Luca's website. It consists of entry, task, verification and exit criteria. (Highsmith, 2002)

The develop an overall model contains according to Highsmith (2002) seven Tasks:

- Form the modeling team
- Domain walkthrough
- Study documents
- Develop the model
- Refine the Overall Object Model
- Write Model Notes
- Internal and External Assessment

2.2.3.2. Build a Feature List

For identifying features, the team containing the chief programmers will divide the domain into subject areas. These are divided again into business activities and then into steps which constitute the features.Highsmith (2002) defines features as something that brings value to the client. Most of the features should be implemented in one to ten days. (Highsmith, 2002)

These are according to Highsmith (2002) the following tasks for this process:

- Form the Features List Team
- Build Features List
- Internal and External Assessment

2.2.3.3. Plan by Feature

In this task, the focus lies on producing the development plan. Therefore, the project manager, the development manager and the chief programmers are getting together and are going to prioritize which features will be developed. Highsmith (2002) states that the planning contains a number of factors such as complexity, risk and client-requirement milestones. The Features are assigned to the chief programmers and then these chief programmers assign developers for the class ownership.

These are according to (Highsmith, 2002) the following tasks for this process:

- Form the planning team
- Determine the development sequence
- Assign feature sets to chief programmers
- Assign classes to developers
- Self-assessment

2.2.3.4. Design by Feature

Each chief programmer is assigned in “Plan by Feature” to some features. He or she will then choose one (or even a set of features) which consists of the same classes. These sets are called Programmer Work Package. The chief programmer then starts to form his feature team, by identifying the class owners. The class owners will start developing sequence diagrams for the features. With the help of these diagrams, the chief programmer will refine the object model. After that, the developers will start writing classes and method prologues. The chief programmer will create a design package consisting of all the diagrams, classes and methods. (Highsmith, 2002)

These are according to (Highsmith, 2002) the following tasks for this process:

- Form feature team
- Domain walkthrough
- Study the referenced documents
- Develop the sequence diagrams
- Refine the object model
- Write class and method prologues
- Design inspection

2.2.3.5. Build by Feature

Developers will start implementing classes and methods according to the design package. The code is then tested and inspected by using “Unit testing”. (Highsmith, 2002)

These are according to (Highsmith, 2002) the following tasks for this process:

- Implement classes and methods
- Code inspection
- Unit tests
- Promote to the build

2.3. Extreme Programming (XP)

XP is a software development framework which was defined by Kent Beck. XP contains values, principles and practices (figure 7) (Sandhaus et al., 2014). According to Schuh (2005) XP is unlike other agile methods, programmer orientated. This means that the developer can occupy any role in XP. According to Beck & Andres (2004), a software development style focuses on clear communication and teamwork. Differences are that it contains short development cycles, an incremental planning approach, automated testing written by programmers, customers and testers. Close collaboration is one prime value of XP.

2.3.1. Values

Communication: Without good communication, most aspects in software development would not work. XP highlights that good communication will automatically lead to good software. Problems which cannot be solved alone due to a lack of knowledge can be solved by consulting a more experienced programmer (Beck & Andres, 2004).

Simplicity: Creating a software which is simple but which solves complex problems at the same time is hard. It is even harder to create simple solutions, which not only work for some but for all problems. XP embraces to think about how it is possible to find a simple solution for problems, instead of solving it somehow without thinking much about it. (Beck & Andres, 2004)

Feedback: Developing a product is not a straightforward process. Requirements or the architectures of systems may change over time. Change is needed; therefore, feedback from the customer is also needed. (Beck & Andres, 2004)

Courage: According to Beck & Andres (2004), courage means to perform effective action in the face of fear. This means that depending on the problem, somebody from the team must have the courage to do the right thing instead of rushing for a solution.

Respect: Without respect XP will not work. Beck & Andres (2004) state that disrespecting other team members or the project itself will lead to major issues and failure.

2.3.2. XP Practices

Beck & Andres (2004) define practices that should be implemented and adapted according to particular needs. Following all the statements will lead to quicker development while it also improves the quality at the same time.

Sit Together: The whole team should sit together in an open space, while having enough space for privacy and being close together for open communication. (Beck & Andres, 2004)

Whole Team: All qualifications that are necessary for succeeding with the project should be there already. Depending on the qualifications needed the team may change; people can leave and people can join. (Beck & Andres, 2004)

Informative Workspace: The environment should tell team members or interested people what problems might occur and how the project is progressing. This can be accomplished by using flipcharts or story cards. (Beck & Andres, 2004)

Energized Work: Do not overwork yourself. Work as much and as long as you can be productive. Working too much without being productive is just wasted time and may affect the next day negatively. (Beck & Andres, 2004)

Pair Programming: Instead of writing a code on your own, XP suggests writing codes in pairs;

sitting together at the same machine while one is programming and the other one is thinking about good solutions or looking for potential problems. Programming together also leads to knowledge transfer. Less experienced programmers are able to acquire new experience this way. Pair programming does not mean that you always have to program together. It is still possible to program alone. (Beck & Andres, 2004)

2.3.3. XP Process

The XP process is defined by the twelve core processes, these core processes are integrated into the XP process with its meetings, events and their interactions (figure 7).

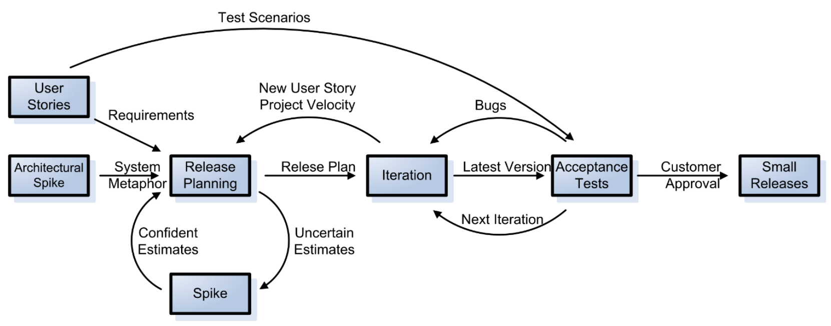


Figure 7.: Extreme Programming process (Schindler, 2010)

Planning Game: In the planning game, the planning is done for the next release. While this is a more detailed plan, a general plan is made at the beginning of the project defining on the overall functionalities and what should be realized in which release. (Schindler, 2010)

Small releases: Means to develop a small increment which provides a business value and to gain quick and accurate feedback from the stakeholder. (Schindler, 2010)

Metaphor: Metaphor portrays the product work in an overall way. It provides a view on how the functionalities work together and what their purposes are and how they correspond with the architecture. (Schindler, 2010)

Simple design: Means that the product must be designed in a simple way. It must meet the needs but nothing more than that. A simple design based on XP guidelines entails the following points: pass all available tests, should not be a duplicate, must be clear and consistent and must contain the minimum methods, classes and modules. (Schindler, 2010)

Test first: Means that test are written before coding.

Refactoring: Simplifying and improving the code without changing its functionality. This should make it easier to understand.(Schindler, 2010)

Pair programming: Instead of working on a code alone, a second programmer should work on the same station and code. This practice can be used for education and for finding bugs. (Schindler, 2010)

Collective Code Ownership: Everyone is responsible for the code and therefore everyone can make changes for any part of the code (Schindler, 2010).

Continuous integration: Every developer must work on the newest version of the code and every change must be integrated to the current build. All the previously and newly added tests must work with the newly integrated code. Whenever problems occur, the code must not be released into the versioning system until the problems are solved. (Schindler, 2010)

40-hour week: Is not a fixed amount of working hours, but it rather provides a guideline for not to overworking oneself. Compared to working over time without making any progress, it is better to work less but efficiently. (Schindler, 2010)

On-Site customer: This means to have a constant communication channel with the customer, in order to gain quick feedback from the person who will use the product. (Schindler, 2010)

Coding standards: Are rules and standards for styling and formatting code. The code is easy to read and to understand. Each developer must agree on this coding standard to ensure consistency throughout the project.

2.4. Comparison of agile software development methods

The table 1 shows a comparison of the choosen agile software development methods. With this it is easier to see the differences and similarities of each development method. Agile development areas such as meetings, artefacts and roles, can be compared.

	FDD	**XP**	**Scrum**
Approach	Iterative	Iterative, Incremental	Iterative, Incremental
Focus	project management process	development process	project management framework
Communication	more written	more verbal	more verbal
User centric	involved in reporting	involved during development process	involved through development owner
Sprint length	2-14 days	up to 6 weeks	2-4 weeks
Information exchange	via documentation	daily meetings	daily meetings
project size	large scale projects	small and simple projects	large and complex projects
Main practices	Object modeling,development by feature, Unified Modeling Language (UML) diagram	pair programming, simplicity, communication,test driven development	Meetings
Parallel feature development	yes	yes	yes

Table 1.: Comparison of agile software development methods

3. Agile Firmware Development

This chapter provides a comprehensive overview of how agile firmware development can be achieved. Additionally, problems which may occur during adopting agile methods will be discussed.

A literature review on agile firmware development was carried out in order to get information, about what firmware development means and how agile software development methods could be adopted to firmware development. Additionally, best practices of companies who already use agile firmware development were searched in order to have a practical view on challenges and opportunity in using agile methods. The results of the literature review will be used to help understand what must be considered in order to adopt agile software development methods in firmware development and what challenges other companies faced and how they could overcome them.

3.1. Definition of Firmware Development

There are many different definitions of what firmware development is. Firmware is software that runs on electronic devices and it is saved on small memory, such as an erasable programmable read-only memory (EPROM), read-only memory (ROM) and flash memory. Firmware provides low-level control of hardware parts for electronic devices. This means that firmware developers cannot only consider software related aspects but they also must consider hardware related aspects. Firmware developers need to consider certain challenges which must not be considered when software is developed as usual. Challenges, such as limited power and memory space resources, force the developer to develop in the most saving way possible. Since not every electronic device has the ability to update its firmware, it is crucial that the firmware is tested thoroughly in order to run without any error in a longer period of time.

Shen et al. (2012) describe six main characteristics for firmware/embedded development.

Skills: Embedded software developers are usually high skilled on the technical level on software and on domain level. However, these developers are not trained to be software developers, but rather started programming because of their needs. (Shen et al., 2012)

Hardware dependencies: Firmware runs on hardware and therefore faces restrictions from hardware such as limited power, memory space and new technology. Hardware is usually developed concurrently with firmware, therefore it is not available until the end of the project, which makes it hard to test the firmware. Shen et al. (2012)

Competitive pressure: As in software development, firmware also has to be developed quickly while lowering the cost to be competitive against other companies. Shen et al. (2012)

Limited resources: Firmware has some resource constraints such as limited memory space, processing power and execution time. Shen et al. (2012)

Changing Systems: If the hardware changes, the firmware also has needs to be adapted to the change since firmware needs hardware. Changing requirements or new developments in hardware also causes changes. These changes also need to be adapted by the firmware. Shen et al. (2012)

Performance: Some fields demand high performance requirements from the firmware, such as safety issues (Shen et al., 2012).

Kaisti et al. (2014) show some similar key indicators for firmware development:

System Level Documentation: In case of agile development, documentation should be kept at a minimum. While this is sufficient for software development, it will not work for embedded development. Embedded development requires system level documentation, as there are many different teams involved and different requirements with strict standards, which require documentation. (Kaisti et al., 2014)

Hardware-Software Interdependencies: Hardware and firmware are tightly connected. Testing together requires that each of them are on a certain state of maturity. While hardware requires long development cycles, firmware can be developed rather quickly. Due to new manufacturing processes, simulation tools and off the shelf components hardware development cycles can be reduced, but will not reach the cycle length of developing software. (Kaisti et al., 2014)

3.2. Adopting Agile Methods to Firmware Development

Most companies, which aim to implement agile development methods for firmware or embedded development, choose existing agile software development methods and adopt them to their needs. In most cases, they use Scrum or XP and adopt them. These methods are commonly used for software development and therefore problems occur if these are used for firmware development. Not all aspects can be implemented easily.

Papers and studies shows how companies tried to implement agile methods to firmware development. Mostly these studies describe their results and how they overcame problems in the adopting phase.

In a literature review of Kaisti et al. (2013), articles are collected in which agile methods are applied to firmware development. They contain surveys in which a positive impact on implementing agile methods in an embedded world is shown. It is stated that in the embedded field, methods such as XP and Scrum are mostly used. Depending on which of these methods are implemented, different challenges may occur. In one study, a new method was created instead of using an existing one. The basis of this method was XP. Some practices of XP were taken and modified. The practices were: test-driven development, refactoring, simple design, pair programming and continuous integration.

Kaisti et al. (2013) state that problems that occur from adopting agile methods to embedded development. Due to lack of support of the software tools and the importance of domain, knowledge hinders the practice of shared responsibility. Moreover, the main drivers are mostly changing requirements from hardware and firmware.

A new paper of Kaisti et al. (2014) rather maps the agile principles than a method to the firmware/embedded world. Delivering a valuable software as early as possible is one of the main principles of the agile principles, but it is also possible to measure such “valuable software” in an embedded world. Kaisti et al. (2014) claim that instead of just delivering a working system, which could last too long because of hardware and firmware development, a better principle would be the demonstration of progress. This can be a demonstration of a proof of concept; a documentation describing the system level design. In later states of the project it can be an actual prototype. The development cycle should be stretched in order to fit the development of both, but usually four weeks should be sufficient.

Delivering a working prototype frequently is also a serious challenge in embedded development. Here the firmware cycle depends strongly on the hardware development cycle. A possible way is to deliver a whole combination of hardware and firmware. Firmware functionalities can be demonstrated even if the hardware is not finished. (Kaisti et al., 2014)

The cost of change is quite expensive in an embedded environment. Changes that are made later can lead to an increase of cost and to a delay of the release date. A solution to this issue can be adding a generality into designs. Later changes are more expensive than creating a general design. New requirements could then be handled as an incremental design changes. (Kaisti et al., 2014)

The self-organizing team is an important principle, as it requires the team to be responsible for developing and decision-making. While this applies to software development, there is a difference in embedded development. It consists of more than just one team and therefore requires some top level in order to cooperate between different teams. The communication between the team members is highly important, even if they are located in separated locations. With modern tools, communication can be established quite easily. (Kaisti et al., 2014)

Aiming for simplicity and optimization is important in agile development, as designing an optimized system in an embedded world could lead to problems if new requirements arise. A good way would be to implement a modular and general platform design. Here Kaisti et al. (2014) try to redefine this principle as balance between simplicity and generality. (Kaisti et al., 2014)

Refactoring and trying to reflect on how to become more effective, is not particularly easy in an embedded system. All these aspects should be reinterpreted. Trying to reflect on how to be more effective can lead to minimized unnecessary documentations and it could also lead to avoiding misinterpretations. Understanding how everything works also improves the methods on how each team works. (Kaisti et al., 2014)

3.2.1. Use Case: Daimler-Chrysler

In this case Manhart & Schneider (2004) describe how Daimler-Chrysler adopted practices from agile methods. The software engineering department developed embedded software for buses and coaches. However, due to customer specific add-ons and features, which were added late during the process, the department faced difficulties. In most cases these features need to be implemented quickly. Instead of weeks or months, they should be implemented in hours or days. The main issues for developing embedded software were:

- Many customer specific functionalities with high time pressure
- High degree of individualization for buses
- Software runs on programmable computers
- Customer demanding top quality

Producing a customer specific bus requires good time management. A delay of parts of the buses causes high economic problems. Therefore, the software will be uploaded even if it is not entirely finished. Later changes of software or functionality during the production leads to delivery delays. Instead, a special upload is processed in which the software is uploaded. However, this process has high costs and it is risky. Manhart & Schneider (2004) discuss even more challenges, such as unrealistic planning, unavailable experts, changing priorities and requirement misunderstandings. (Manhart & Schneider, 2004)

In Manhart & Schneider (2004) they analyzed their software development process based on the new goal driven improvement philosophy. Their goals were to increase customer specific software based functions and to have a zero bug policy. Based on that, they found difficulties in their current work situation, which should be solved by applying some basic principles of agile practices:

- More work, but less development staff
- More frequent late changes
- Top quality goal
- Failure leads to late delivery

Based on that they chose some principles and defined activities.

- Implement features with high value first
- Find differences between specification to the product as automated as possible
- Find them as soon as possible
- Fix them as soon as possible

Because of these principles and because of the goal driven process they chose the practice test first and the unit test in which they create unit tests before developing. With these tests, they hoped to achieve a quicker test of simulation, improve the quality of assurances, reduce differences between module and specification, reduce the effort for quality assurance, start to describe tests before coding and to focus on real requirements first. (Manhart & Schneider, 2004)

In their paper, they did not mention their results of the implementation, because they wanted to share their experience with problems in implementing agile methods as quickly as possible. According to Manhart & Schneider (2004) it is not possible to implement an agile method without changing it to their needs. However, instead of rushing and trying to implement the whole method, Manhart & Schneider (2004) recommend implementing it step by step in order to ensure that all the staff agrees with it.

3.2.2. Use Case: Mass-Produces Embedded Systems

A paper of Eklund & Bosch (2012) presents several cases from the Volvo Car Cooperation (VCC) and for them they present a method of introducing agile development in the embedded system's world. Factors for implementing agile methods in an embedded software environment with a stage gate process for hardware development were considered. The method was then validated with three more cases. Only one is finished at the time presenting the paper.

The first three use cases describe the domain of mass produced embedded systems. The purpose was to implement a new distributed software architecture, introducing a new architecture maintenance process and project of an infotainment system. In these cases, issues were found which had to be solved. They also asked themselves in how far these problems were relevant for their research on introducing agile in embedded systems. Issues found in the first case were a more integrated functionality between different components. Software was programmed by different teams that needed synchronizations. In the second case, the found issues were related to the development process. Because of this process, design documentation became quite complicated to understand. Additionally, integration efforts for verification and validation became overwhelming, while being easy in adding new features. In the third case, a new infotainment system had to be developed, while each software development had been outsourced. Issues were that they had a rather complicated interface between micro controllers and therefore between suppliers. The setup of the team changed in the middle from component based development to feature based development with cross functional teams. There was no overview of realized features. Moreover, the sprints were too short with less verified features after each sprint. (Eklund & Bosch, 2012)

Based on these cases they characterized the domain of mass produced embedded systems. According to Eklund & Bosch (2012) they found that these systems have:

- Deep integration between hardware and software for important parts of the functionality
- Strong focus on manufacturing aspects of the product in the development
- Strong supplier involvement
- Some parts realize safety critical functionality

A model was developed which shows the interaction between the mechanics, hardware and system development and an agile/traditional software development. These interactions are also necessary for agile teams, while having the problem that not the whole product is agile produced. They define categories for interaction between the embedded software and the product development such as requirements, project gates, integration and validation and delivery. For these categories, they evaluated what has to be implemented in order to achieve agility. To do this, they chose XP and Scrum. These measures, which had to be implemented, are divided into the pre-game phase and the activity phase. (Eklund & Bosch, 2012)

Requirements: For the pre-game phase, they wanted a product owner which is responsible for prioritizing requirements and who values the agile principles. In the activity phase, they wanted to have a greater acceptance on an incremental growth instead of the typical waterfall model, while using a broad area of different project tools at the same time. Not only should features be added to the backlog but also quality attributes and architecture solution should be applied. Therefore, the architect has to interact more with the development team during the project. (Eklund & Bosch, 2012)

Product Project Gates: Here the focus lies on the interface between software development team and the full product project. For the pre-game phase, a clearer structure should be implemented in order to improve the handling of change requests. The project manager should be present at the sprint demos, as he or she should be able to see how the project is developing. A connection should be established between agile roles and methods to existing roles and functions of the organization. In the activity phase milestones, artefacts and gates should be included in the work of the agile software team. (Eklund & Bosch, 2012)

Validation: Here the focus lies on the interface between agile software development and the validation of the product. For the pre-game, they wanted to align the pulses of the Original Equipment Manufacturer (OEM) and the subcontractor's sprints. In the activity phase, all the technical standards have to be verified and validated and the quality requirements have to be checked. For producing new deliverables in time and to test them, a system anatomy should be defined to check technical dependencies between implementations in each sprint. Defining interfaces to sub-systems (synchronized backlog). (Eklund & Bosch, 2012)

Internal Activitys: For the pre-game phase, it is crucial to ensure that the entire platform used by the development team is available. This means providing resources to all of the roles that participate in producing the product. Moreover, this means establishing a connection between OEMs feature definition and the testing teams and the subcontractors' development team. In the activity phase, it is stated that agile development should only be introduced when people are eager to try it and are eager to implement a frequent refactoring and clean up as an activity. (Eklund & Bosch, 2012)

Method Validation

For validation of their method, they used it again on three cases. While just one case is finished, the others are still in progress.

In the first case, they developed an infotainment system again, based on an open platform. The goal was to check whether it was possible to develop a feature driven way with short lead times (four-twelve weeks). This small lead times where accomplished by using Scrum. The infotainment system was tested in a real environment. It was installed in simulators and in test cars. The project contained two different teams. Each team had their own backlog and design documentation but shared them over a software that was separated from the hardware and platform development. However, the problem at this point was that it was not based on a release of a new car. This means that there were no demands for gate reports. The team members met every two weeks for half a day. The focus was not on testing or quality assessment. Based on their model they chose for the pre-game phase: a product owner who had a clear structure of the goal process and who connected agile roles and methods to non-agile ones. He also had a mapping between OEM and the subcontractor and aligned the OEM pulse and the subcontractors' iterations. As for the activity phase, they implemented a gradual grow of requirements while interacting with existing tools. They defined interfaces between subsystems and implemented a frequent refactoring. (Eklund & Bosch, 2012)

The second case is about developing a climate control software in-house instead of outsourced. As an agile method, they used Scrum. The team consisted of nine persons and a Scrum master. The product owner was a person from the interior department, with cooperation from one person from the product-planning department. The developed software was used on hardware that had been developed from a hardware supplier. Problems here concerned setting the requirements into a suitable form for the backlog. In this case, most of the method principles for the pre-game phase and the agile phase were adapted. They searched for a dedicated product owner, which was a challenge. The product owner is a person outside the development team. Clear structures, processes and platforms were established. For the activity phase, they wanted to have a gradual growth of requirements, while interacting with existing tools. There were also some nonfunctional requirements in the backlog. The development team adjusted sprint schedules in order to align the schedule for integration of the complete systems. An coach was involved in this case. This helped them to achieve a good level of agility. (Eklund & Bosch, 2012)

The last case was about producing a next generation infotainment system for future car models. The same department was involved as in the third evaluation case. Here the team already had some experience in agile methods. Requirements were defined on a feature level instead on a design or implementation level. The development schedule of the ten teams of the department was set with the subcontractor in a six-week iteration, in order to meet their schedule. In this case they did not start a sprint at the point their paper was written, therefore there is not much information. However, in their case they tried to implement all of their principles from their method. (Eklund & Bosch, 2012)

3.2.3. Use Case: Intel Cooperation

In the paper of Greene (2004), their experience of using an agile approach for developing firmware for an Intel Itanium processor family is explained. As in other firmware projects, their development of firmware had to also be aligned with the development of the new processor family. Because of the high uncertainty and the changes they had to make while developing, they decided to use an agile approach. Their new Intel Itanium processor family should consist of a 64-bit architecture. This architecture should be able to implement behaviors completely in hardware or in combination with firmware (Processor Abstraction Layer (PAL)). PAL should provide an interface across all the Itanium processor family designs in order to enable abstract implementation of specific features.

Team and development: Their firmware team consists of seven developers and one manager/technical leader who is part of a larger processor design team of over 100 hardware developers. Two of their developers are located at different places. Their hardware team consists of architects and different designers who have worked many years on the first phase of the design. The PAL firmware team is developing firmware to test the processor and to implement features. Changing hardware mechanism leads to a demand of change in the firmware. While this is sometimes acceptable, they have problems to develop the code until the silicon is ready. The next phase entails of testing the processor and trying to fix changes by changing their firmware. (Greene, 2004)

Problems

As in many other firmware teams, they also had some problems with developing the hardware simultaneously. Their main problems were to follow the detailed scheduled plan. Firmware team members were too specialized in their domain, changes were made without developing tests, and there was an overly optimized assembly code and inconsistencies concerning the coding style. It seems that these problems may arise from a lack of software development methods. Therefore, they implemented XP. Concerns regarding the evaluation from the code, using pair programming, privacy and testing before coding brought them to reconsider whether these could be applied into their project. In the end, they combined different practices from Scrum and XP to meet their needs. (Greene, 2004)

Scrum Practices

From the Scrum method, they took the following practices:

Sprints: It seems to be the case that doing a thirty-day sprint provided a good granularity for planning their requirements. However, some developers did not approve to this and wanted more visibility beyond thirty days. (Greene, 2004)

Sprint plan meeting: Instead of implementing half-finished features, they used these practices to implement features that had been entirely finished and tested before only. (Greene, 2004)

Daily Scrum: Team members got information on what the others did the day before, but after some time they changed to focus on "what are you going to do today" and "what are the roadblocks". Time spent on these practices was reduced to a minimum.(Greene, 2004)

Sprint review(retrospective): Was quite helpful at the beginning but after solving many sprints they had not much to talk about due to the close interaction. (Greene, 2004)

XP Practices

They found out that XP was quite effective for bringing software methods to people with hardware backgrounds. The practice mostly used was unit testing. As more tests were created, because automated tests were used, the quality of the code increased. (Greene, 2004)

Simple Design: Optimized assembly codes lead to complex maintenance. Moreover, group ownership and pair programming lead to a more simple design. (Greene, 2004)

Unit Test: Creating tests for coding leads to a shift in mindset. By developing tests first and improving the quality of the code itself, less bugs were produced. The Problem here, however, was that there was no "Assembly Unit" testing framework. Therefore, they had to build one on their own. It was possible because of their framework to test components in their low level. This meant that they were able to implement them without any concerns of having bugs. Scripts for checking the results were also developed. Overall, they carried out 1300 tests, which ran two hours per night and some ten minutes tests for the developer. (Greene, 2004)

Refactoring: Improving their designs by simplifying it and improving the codes maintainability and flexibility lead to a decrease in legacy code. (Greene, 2004)

Pair programming: It is useful to find bugs and to transfer knowledge to less experienced team members. In their paper, they state that the initial coding from the assembly code was quite useful. However, people wanted to split afterwards in order to solve problems more quickly. Instead of forcing them to stay together the entire time, they let them decide on their own when to program together. Programmers paired together at the beginning then split up and got back together to check their code with the help of others. Transferring knowledge was not especially effective, as not everybody is able to know everything. This is particularly true in complex environments, but they had the idea that one day of a week each developer cannot touch anything in their expertise range. By doing this, they learned much more. Team members that were located in different cities got involved by using audio bridges or virtual computing networks for pair programming. (Greene, 2004)

Collective Ownership: It is hard to implement collective ownerships in an environment with experts in special fields, but instead of sharing all the ownership, acceptance of the fact that developers can improve their code was implemented. (Greene, 2004)

Continuous Integration: Due to their modular architecture and their concurrent version-system, developers were forced to integrate changes from the repository to their workspace to change something. (Greene, 2004)

One-Site Customer: Due to the PAL firmware architecture there is no external costumer. Prioritizing was done by the creator of the paper. However, the hardware design team was requesting firmware changes. As a result of the collocation of both teams, close interaction was possible and it was therefore easy to discuss changes. (Greene, 2004)

Sustainable pace: Hard deadlines required the team to do some extra hours sometimes (Greene, 2004).

Coding Standards: A standard was created for Itanium assembly, but sometimes workarounds were done which were not refactored (Greene, 2004).

3.3. Summary

According to the use cases, implementing agile software methods in a firmware/embedded environment is possible. However, there are restriction that do not occur in a pure software development. Most of the agile methods are developed for pure software. Therefore, the practices from the used methods needs to be adapted or not all of them should be used. The cases show how companies just implemented agile practices and not a method, such as in Manhart & Schneider (2004), and some cases in which they took two agile methods and combined them to one agile method Eklund & Bosch (2012); Greene (2004).

	Daimler-Crysler	**Mass-Produced Embedded Systems**	**Intel Cooperation**
Author	Manhart & Schneider (2004)	Eklund & Bosch (2012)	Greene (2004)
Sector	Buses and coaches development	Infotainment Systems	Chip development
Agile method	Only some agile practices	XP, Scrum and Stage-Gate	Some practices from XP and Scrum
Challenges	customer specific functionalities, high time pressure, customer demanding top quality, delivery delays, unrealistic planning, changing priorities	complicated integration between hardware and software, complicated documentation, agile and traditional development, outsourced development	developing firmware/hardware simultaneously, following detailed plan, specialized firmware developer, overly optimized code, no tests written
Benefits	quicker tests of simulation, improve quality, reduce effort, better focus	short lead times, coach lead to better agility	shared information, planning only features which could be finished in time, group ownership, better code quality
Takeaways	test first	find a good Scrum master, own backlog for each team, alignment of different iterations, search for a dedicated product owner outside of the team, plan on feature level	adopt practices to your needs, daily Scrum helps to have a common ground, group owner ship and pair programming can lead to a more simple design, refactoring improves code maintainability

Table 2.: Summary of literature review for agile firmware development

Because Scrum is rather seen as a management framework which helps to plan a project but does not help with how to develop and what kind of development tool should be used, it is used in combination with XP. XP is more developer centric and therefore provides some practices and methods on how to develop; such as testing before coding or using one standard. A combination of both of this cases lead to an improvement of quality and to an improvement of the development cycle.

All of these cases present positive results in implementing them and only show a few disadvantages. Some agile practices, such as planning a game or a metaphor, were not implemented. Only practices that caused great advantages in the development phase of their product were used.

As in the case of Greene (2004), collocating their hardware design team with the firmware team lead to a more open communication and effective discussions for design changes and code improvements. Prioritizing features and implementing code only then when they are entirely finished and tested lead to an improvement of quality in the code and prevented the developer from being distracted by other requirements that were not in the sprint. Daily sprint meetings caused all the team-members to share their current results and problems they were facing. Everybody knew what the others were doing and could help them if they faced some difficulties.

For creating an iteration based cycle Eklund & Bosch (2012) state that it is crucial to design the firmware and the hardware in a more general way. For testing the firmware, Eklund & Bosch (2012) show that at first tests were done in a simulation rather than on the hardware. However, after the silicon was ready they tested it on the hardware. Here Kaisti et al. (2014) recommend to use a rapid prototyping or to test on an evaluation board in order to identify problems as soon as possible.

In Eklund & Bosch (2012) sprint cycles were mostly adapted from the software methods, but at the same time they also tried to align it with the hardware team. Therefore, a good communication is vitally necessary. There is no standard on how to align these sprint cycles but due to up-front development or rapid development it is possible to lower the sprint cycles of hardware and therefore it is also possible to align both of them. It is also possible to not present a prototype after every week but rather the result on a simulation or a documentation on the design itself (Kaisti et al., 2014).

To conclude, it is possible to adopt such methods in an embedded environment. However, the people need to be very eager to try it and it should not be implemented as quickly as possible but rather in a slow pace and perhaps one method at a time, to ensure that all the team members get familiar with it (Greene, 2004; Eklund & Bosch, 2012). A summary in form of the table 2 was put together in order to have a better overview of the use cases and the takeaways.

4. Agile Hardware Development

This chapter provides an overview of how agile hardware development can be achieved. Furthermore, problems with adopting agile methods will be discussed.

A literature review on agile hardware development was carried out in order to gain information about the meaning of hardware development. This literature review also provides information on how agile software development methods can be adopted to hardware development. Furthermore, best practices of companies which already use agile hardware development were searched in order to have an authentic perspective on challenges and opportunities in using agile methods. The results of the literature review will be used to help understanding if it is possible to use agile software development methods in hardware development, what must be considered in order to adopt agile software development methods in hardware development and what challenges other companies faced and how they handled them.

4.1. Definition of hardware development

Thompson (2015) define hardware as electrical or electro-mechanical devices, which often contain firmware or embedded software. Developing hardware is the same as developing software, in the end it is a process for developing a design that is intended to be produced.

Agile hardware development tries to adopt methods from agile software methods to hardware development. The reason why agility is adopted into the hardware development is to reduce uncertainty, reduce development costs and to increase customer satisfaction and at the same time shorten lead-time. Instead of a stage gate model or waterfall model, methods such as Scrum and XP are adopted. While this is quite easy to achieve in software development, there are difficulties to solve beforehand with hardware. (Schuh et al., 2016)

There are similarities and differences to software development and therefore processes for software development cannot be adopted without some changes.

According to Thompson (2015) similarities are:

- There are interactions between users, products and outputs.
- They consist of functional and nonfunctional requirements.
- They are complex.

Differences are according to Schuh et al. (2016):

- Changes are more difficult than for software.
- Changing costs are high for hardware.
- Hardware consists of physical components that cannot be refactored after being produced and cannot accrete new capabilities that requires hardware changes.
- Hardware designs are constrained by standard parts.
- The design of hardware is driven by architectural decisions, more architectural is done up front.
- Fewer tests but with specialized and expensive equipment.
- Hardware tests are done by the creator.
- Hardware must work for a range of time and under different conditions.
- Longer lead times for acquisition for specialized hardware components.
- Costs of hardware development increase towards the end of the product cycle.
- Changes done in hardware can lead to higher costs due to postponed shipping schedules, sink costs.

4.2. Adopting Agile Methods to Hardware Development

In the paper of Thompson (2015) a research was conducted on how Scrum processes can be adopted. Because of the differences between software development and hardware development, Thompson (2015) describe changes made in the Scrum process to achieve agility in hardware development.

User Stories: Stories are tasks that should be achieved in one sprint. However, instead of using just “hard stories”, Thompson (2015) recommend to also use “soft stories”. Only if the story cannot be developed and tested in one sprint, there is the possibility to set it as a “soft story” which could be developed in two sprints or more.

Sprint length: Due to several interviews and from personal experience, Thompson (2015) recommends a sprint length from two to eight weeks, depending on the difficulty and the possibility to split tasks into deliveries which could fit into a two weeks sprint. Variety in length should not be an option, instead it is recommended to use a fix length which should work for all the tasks. Using an appropriate length can reduce the risk of not delivering after the sprint and it guarantees having a better flexibility and a better sprint tracking.

Release planning: Planning to deliver a product after a release cycle is hard for hardware development. Not only does the hardware need to be considered but also the software needs to be considered. However, according to Thompson (2015) planning a release date is crucial so that a reliable plan can be made. Additionally, it leads to having a reasonable concept of the product.

Sprint deliverables: Hardware cannot always produce a deliverables with new features after each sprint. However, deliverables can be produced and tested at each release cycle. Thompson (2015) claims that it is possible to create a prototype in the first sprints and then try to iterate the prototype until the requirements for the product are met. The focus of each sprint may vary according to Thompson (2015), but it also needs to be considered that more variation also leads to complex planning. Therefore variation should be kept at a minimum.

What is also crucial for adopting agile methods is having highly motivated team members and a high density of communication. Team members should be located in the same area and all qualifications and all skills which are necessary for designing, developing and testing the product should be available. Another crucial aspect is to ensure good communication with the suppliers. It is vitally necessary to involve them into the planning, in order to ensure that developing hardware and delivering the parts can be accomplished in a short period with less costs. (Schuh et al., 2016)

In their paper, Schuh et al. (2016) describe what is necessary to adopt agility into a product development. They split it up into three categories:

Process: Because of shorter development times and iterative cycles, the product has to be split into small development tasks that can be realized in a short time. Iterative testing is then done on parts or on the prototype. For developing iterative prototypes components, the material needs to be delivered in a short period, otherwise the procurement would be time consuming and this would increase the costs. As a result, having a strong communication and a better integration of the supplier is necessary. The disruptive network approach could constitute a solution. This approach requires a high integration of all partners. (Schuh et al., 2016)

Project Team: Instead of having an indirect communication, a more direct communication is necessary to achieve more agility in product development. Therefore, team members should be located in the same place or in the same room. For hardware development, it is also crucial to involve the purchaser into the development of the product. This needs to be realized in a way that he or she is dedicated to one project. This leads to higher technical insights. For having agility and flexibility in procurement, higher prices are paid. This should be discussed with the purchaser in order to prevent delays. (Schuh et al., 2016)

Product/Technology: Due to iterative development cycles, frequent testing and more changes to the product are possible. While this is good, problems occur with buying parts that are needed beforehand. New suppliers are needed which can deliver in a more flexible way. A way to reduce costs would be buying components that are already on the market. This means that an own development of components is not needed. (Schuh et al., 2016)

An article of Backblaze Inc. (2015) addresses the same challenges as stated above but proposes different solutions. For the frequent release principle, it is easier to not create a product after each sprint but rather to deliver virtual simulations. This would solve many problems in agile hardware development such as the cost problem of changing the design. Here it would be easy because there are no physical components and it would lead to more modularity. Another solution for the same problem would be to focus on a particular component for each sprint. Instead of designing the whole project, only one component is designed and later in the release cycle, a sprint will be defined for the integration of all components. (Backblaze Inc., 2015)

The master thesis of Reynisdottir (2013) analyzes the differences between hardware and software and this thesis examines what aspects need to be changed in order to adapt agility. He claims that the main difference is that hardware deals with physical objects and therefore has some constraints. With hardware there is a waiting time for ordering parts and testing cannot be done as frequent as with software. Moreover, the construction of the product with all the components takes some time. Therefore, he states that it is rather unlikely to have a shippable product at the end of each sprint. However, he states that it is possible to use mock ups, computer-aided design (CAD) models, simulations or rapid prototyping to create prototypes that can be delivered. Rather than trying to produce a working prototype at each sprint, it seems more reasonable to produce one at each release cycle, while the goal of each sprint could be to reach a specific goal.

Additionally, if we think of having an overall team, it seems not practical on products that contain hardware, software and firmware. On the contrary, it seems much more practical to use different teams with Scrum and to have a Scrum of Scrums. The sprint length could also vary in order to fit their needs only if it is justified. (Reynisdottir, 2013)

4.2.1. Use Case: John Hopkins University Multi-Mission Bus Demonstrator

In the case from Huang et al. (2012), agile hardware and software development was used to design two small satellites. The satellites did not use any existing key technology. Instead, a rapid design and development was necessary to allow changes and assessments in the hardware. In this case, requirements are hard to determine because of changes and extensive interactions with the customers. While designing high tech systems takes at least two design cycles to finish and mostly take a third quarter until the end to be frozen, high uncertainty and a lot of changes lead to a high flexibility of the engineers.

Challenges: The challenges for the Multi Mission Bus demonstration was to create a satellite within the CubeSats standardized volume, in order to enable the satellite to launch with other things, to split costs and to use extra available space. Because of the CubeSat program, a new market for commercial of the shelf hardware was created to ensure that anybody who wants to build a satellite could do this. The project uses this hardware to save design costs, but CubeSat has its own engineering standards which does not require the passing of typical tests for launching. (Huang et al., 2012)

Practices: Their focus was to increase functionality in a CubeSat format while balancing costs, schedule and reliability. Because of creating a high tech product, the project had to be highly flexible in order to overcome long periods of uncertainty and changes. The key points to their success were: individual and interactions, emphasis on approaches toward a working system, a collaborative interface with a sponsor, responding to changes in scheduling and tasking in a flexible way. (Huang et al., 2012)

Individual and Interactions: The project team consisted of highly experienced staff that worked in high-pressure situations. Each of them lead a sub-team and each of them had the permission to make decisions on their own and to talk directly to the sponsor. Instead of using a normal space program development organization, they flattened it and created a new one. External experts could be called if needed and dismissed when finished. The program manager reported directly to the head of the space department and therefore was able to implement changes when needed and he could make personal decisions. Decisions that needed some higher ups could be resolved with the program manager. This enables the staff to be quicker with the requirements, while also maintaining cost and schedule. Team members were co-located to enable communications and discussions. Instead of using a traditional linear project flow of National Aeronautics and Space Administration (NASA), they used a modified version creating a non-linear one to be more flexible. They used the concept of build a little, test a little, and learn a lot, which means that instead of having a number of design iterations and many prototypes, small incremental steps were taken and issues were fixed as soon as they occurred. (Huang et al., 2012)

Emphasis on approaches towards a working system: Only one review was funded in order to maximize the benefits of designs and reviewers. This concept was called "only design review". Discussions for the requirements were held verbally, for discussions on the concept and for the features a CAD model or simulations were projected and manipulated in real time. Due to that the presenter was able to answer questions in detail and this also allowed the review team to have more detailed view on the design. Reviewers were selected intentionally. To add value, other non-scheduled peer reviews were held during the program. (Huang et al., 2012)

Documentations were minimized, while all the components drawing were kept and stored under a configuration managed system. The signature list for the component drawings only consists of the originator and the leader. (Huang et al., 2012)

To ensure more flexibility, sources for parts were selected depending on whether they were critical or non-critical parts. Non-critical parts were procured depending on the turn-around and the lowest cost. Parts that required high precision and a high tolerance were made on NASA certified manufacturing facilities. Those facilities gave the engineering staff the possibility to monitor the fabrication process. Costs and cycles could be reduced by using directly the design files to program the machines. (Huang et al., 2012)

Collaborative interface with sponsor: The sponsor of the project was an active collaborator with the development. He participated in all major reviews and provided feedback and inputs in order to assure that what is produced is what is needed. Due to face-to-face meetings with the sponsor, questions were immediately answered and unnecessary costs thereby avoided. (Huang et al., 2012)

Responding to changes using fluidic scheduling and tasking approaches: Co-located interdisciplinary teams were created only for this project. These teams had special interest in schedule, cost and the scope of the project. Daily team reviews were conducted using Scrum boards which contained all the tasks and issues. Adjustments to the highest priority task could be made much easier due to the Scrum board. Responsible team members for tasks could be seen, and tracking of progress for each tasks, tasks dependencies and identifying possible bottlenecks could be done more easily. The highest priories were mostly given to bottleneck issues. (Huang et al., 2012)

Scrum meetings where held quickly and afterwards ad-hoc side discussions could be held in breakout session. The board also helped the part time team members to see the results of tasks and to find the responsible person to gain information. The board helped the team visualizing the current speed of each element progress through the project. (Huang et al., 2012)

Recommendations

Huang et al. (2012) provides some recommendations based on their experience and research throughout the project:

- Use small teams with a direct link to sponsors/customers to keep them well informed by including the sponsor into the team.
- Each lead will have the authority and responsibility for their subsystem, for the interface and interactions with all the other subsystems. The project manager must have the authority to also report to the highest-level organization.
- Use experts from outside the company on a part time basis when needed.
- Co-locate the technical leads, systems engineer, quality/mission assurance manager, and program manager, review all the tasks, issues, cost and schedule every day.
- Present concepts and designs in an interactive design review to uncover issues concerning them.
- Adapt existing processes to the needs of the project. Minimize documentation, but important work must be configuration managed.
- Analyze and test as early as possible to uncover issues.

4.2.2. Use Case: Svenska Aeroplan Aktiebolaget (SAAB) Electronic Data Systems (EDS)

In this case, a Scrum like method is used in a multidisciplinary hardware team. SAAB EDS is a technical company in the defense industry. In this project, three functional lines were involved: power and cabling, mechanical and microwaves. In the entire method, no software competence was involved and this team did not cooperate with any other team. It consisted of three to the project dedicated persons only, with twelve persons involved that also were working on other projects. The three dedicated team members were product owners. One of them served as the Scrum master. At the time this study had been carried out, the project was already in its ninth month (out of 22 months). A prototype had already been built. It consisted of eight parts and each part went through the design, build of prototype and verification phase. (Reynisdottir, 2013)

Method

The project used a Scrum like method. Before that, they had a visual representation of the plan for the project with tasks for each person. They implanted some methods from Scrum such as a sprint of three weeks, no particular training was done for the team and only a one-hour introduction of the method was held. (Reynisdottir, 2013)

Scrum Board: As in Scrum they also used a Scrum board. Their Scrum board was split into four columns. The first three columns were marked red and reserved for the sprint backlog and the last column was reserved for the burn down chart and the next for work items. For the sprint backlog columns, they split it up into a backlog column, a checked-out column and a done column. While the done column was split into two parts, the second part was - when the tasks was done and when the tasks had already been accounted – reserved for the burn down chart. Magnets with the team members' face on it were used to determine which person was responsible for which tasks. (Reynisdottir, 2013)

Daily Stand Up: As in Scrum, they did a daily stand up. One for the product owner in which the Scrum master asked each person about the status of the checked-out tasks. Discussions were interrupted if they were too long or too technical. (Reynisdottir, 2013)

Planning: Instead of planning 100% of the time, they only planned 80% and 20% for extra work. A rough plan was made with the project manager and the product owner. Afterwards it can be corrected for each sprint. (Reynisdottir, 2013)

Work Breakdown: After the first sprints, the work breakdown improved. Instead of having trouble breaking down tasks with ten days estimation, they were able to break down tasks to two or three days. (Reynisdottir, 2013)

Prioritization Meetings: The project manager, the product owner and one or two team members met to decide based on which work items they should plan the next sprint. These items were sent to the team in a list. After that, the team met together to estimate the capacity of the sprint and if the time needed exceeds the limit of the first meeting. Some remaining work items were put into the next box. The team was often not able to finish everything that was planned; 10% were left. As units for estimation, they used days and hours. The capacity was calculated for each sprint and therefore the sprint time varied for each one. (Reynisdottir, 2013)

Changing Priorities: In Scrum, priorities, which were set for stories, should be kept through the whole project, but this did not happen in that particular case. They changed the priorities for the stories between each sprint, depending on how important they were. If some unexpected extra work was done, they used pink notes to explain why they were not able to finish other tasks in time. If some stories became irrelevant, they lowered their priority and put them in the next sprint with a higher priority. Due to the idle times and the waiting times for externals to finish work or to deliver parts, the developer may accomplish other work from the next sprint. If there were none, the project manager tried to find some other work for the person. (Reynisdottir, 2013)

Dependencies and Updates: For estimating the delivery time of the parts, a Gantt chart was used. The new planning feedback about activities that could not be completed was received sooner than before. This lead to an earlier update for the in-house customer and the boss. People felt happier due to the new method. They were more a part of a team now than before. (Reynisdottir, 2013)

Differences to Scrum

Roles: Instead of having stricter roles, they decided to use a product owner as a Scrum master, which is usually not the case. (Reynisdottir, 2013)

Review/Retrospective Meetings: In the previous sprint they did not review or attend retrospective meetings. Instead, they wanted to try it in their current one, at the end of the design phase. The product was verified afterwards. (Reynisdottir, 2013)

Recommendations

Reynisdottir (2013) recommends writing detailed goals for the backlog, in order to find a good and experienced Scrum master. It is recommended to try to improve the communication between the Scrum master and the Scrum team.

4.2.3. Use Case: Marel Garðabær (GRB)

In this study a global provider for advanced equipment and systems for food processing is trying to achieve agility in mechanical hardware development. The embedded software team had already been using Scrum. Both teams had to cooperate in order to develop a single product and must therefore be somehow synchronized. Due to the many successes in using Scrum and because of a new incoming project, an internal decision was made to implement Scrum in the mechanical team. To do this, an external Scrum master was consulted in order to help the team and the project owner was not a part of the team anymore. The author of this thesis has carried out some interviews. Based on the results and based on his observations, he found out which artefacts were not used for Scrum and he made some recommendations in adopting agile development methods. (Reynisdottir, 2013)

Roles

Scrum suggests that the teams should be cross-functional, which means that all the competences needed for finishing the product should be gathered in the team. In their study, the teams were not cross-functional, but the communication and cooperation between the different teams improved since Scrum had been implemented. It needs to be considered that the team in this case was self-organized but still needed help for planning from the product owner, which frustrated the team. (Reynisdottir, 2013)

Product Owner: Here the product owner was responsible for the product value and for the team. However, the product owner did not manage the product backlog, instead the release plan served as a backlog. He also did not manage it. The team wrote almost all the stories. The stories were more technical and solution oriented. The product owner stated that the team should only focus on the project, as at the beginning they were helping other projects with older designs they made before. Representing the stakeholder was also a difficult task, especially when there were requirements that contradicted each other. Moreover the product owner was not completely sure what his role was and what his responsibilities were. (Reynisdottir, 2013)

Scrum Master: Is responsible for Scrum and for the process itself. In this case, the Scrum master was not a certified one and therefore lacked knowledge. At sprint 7, they changed the Scrum master. Serving the team and helping them understanding the rules was achieved by the new Scrum master, but he did not tell externals whether their interactions which the team was helpful or not to maximize the Scrum team's value. He helped the team understand that everybody should be allowed to take any task from the board. (Reynisdottir, 2013)

He did not invest much time in guiding the product owner. However, he should have done that and he should have helped managing the backlog. Additionally, there was only one Scrum master for the embedded software team and for the mechanical team. (Reynisdottir, 2013)

Development Team: At the beginning of the project, the team did not actually know what their goal was and how they should work on reaching it. On the contrary, they solely worked on their tasks. During later sprints, the team became more and more confident in reaching their goals, since they were working on a physical deliverable. (Reynisdottir, 2013)

Here the team was not cross-functional; not all competences were realized in the team. Sometimes guidance from outside was needed. Most of the time the team-members decided who should do what on their own. They only needed some assistance from the product owner or the Scrum master for planning. Because of the nature of hardware, the team could not always deliver a working product increment. They rather tried to reach a goal for each sprint and delivered some designs and a bill of material lists. Mock ups could be another way to deliver something, but therefore the sprint takes more time. (Reynisdottir, 2013)

The team did not have any titles or a shared ownership for the whole product. Instead of having only one team, they divided the team into hardware and embedded software. This is a contradiction to Scrum, but still lead to a better outcome for planning the sprints. Good communication and cooperation is needed for having sub teams. Tasks were done by the team- members themselves, nobody told them how they should do certain tasks. Only one time the product owner had to change some tasks because of some new stakeholder information. The product owner already provided a technical solution, which he should not do. Asking the team for solutions would have been better. Consisting of only four members for the mechanical team, they were small enough to have a good communication and large enough to discuss and to perform significant work in each sprint. (Reynisdottir, 2013)

Scrum Artefacts

Backlog and Items: Reynisdottir (2013) states that they did not have any backlog. Instead, they created a release plan that served as a backlog. The release plan did not evolve and was not managed by the product owner. Release stories were described as a subsystem and as components. The items were not written from a user perspective, they were more technical. Ownership for the sprint stories was kept in the team. They also created the stories for every sprint on their own. Backlog items should be expressed in acceptance and quality criteria and as an analysis design. The team did not agree to this. However, this could have help them to improve. (Reynisdottir, 2013)

Definition of Done: They did not have a real definition of what “Done” is. It was often unclear how to define a general definition for different tasks. (Reynisdottir, 2013)

Burn-down Chart: Instead of having a Burn-down Chart for each sprint, they decided to use only one for each release plan they had made. (Reynisdottir, 2013)

Scrum Board: They used a wall sized Scrum board, to help them visualize what has been done and what they should do next. They also used it to stay up-to date with the progress of the current Sprint. (Reynisdottir, 2013)

Scrum Events

The team felt that the Scrum meetings took too long and were unnecessary. Even if they did not want to, they attended it and started discussions and conversations. This is valuable for finding solutions for technical problems. This might not have happened without them. After moving from a concept design to a detailed design, the planning and daily meetings became shorter. (Reynisdottir, 2013)

Sprint: These sprints were two to four weeks long. Instead of always having fixed sprint lengths, they varied the size of a sprint for special circumstances if it was more practical for the team. The sprint length changed three times (figure 8):

- In Field tests(Sprint no. 7)
- During the synchronizing process with software team sprints (Sprints no. 2 and 9)
- During Vacations (Sprint no. 1 and 8)

(Reynisdottir, 2013)

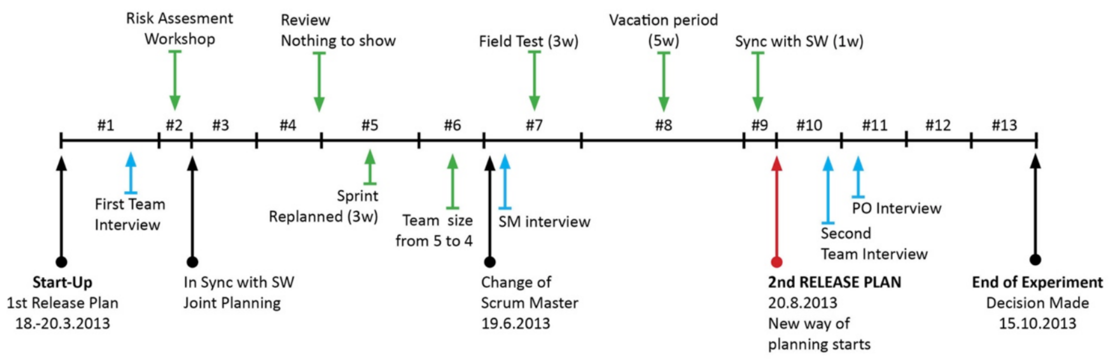


Figure 8.: Scrum Sprint Experiment Timeline (Reynisdottir, 2013)

Reynisdottir (2013) shows a timeline that displays changes in the sprint length, and various events that happened during the sprints (figure 8).

The sprints were held one after another without any breaks and without extending the current one. Only one time in sprint no.5, new information had been gained and the sprint plan was redundant. (Reynisdottir, 2013)

Sprint Planning: Their planning differs from the one explained in the theory. The team often decides on its own what it is going to do in the next sprint. Moreover, no one else than the team and the project owner attends the planning meeting. The project owner only attends the beginning of the meeting and agrees to the plan at the end. Here in this meeting they do not plan "what" they are going to work on and "how" they should do it. This happens in the next meeting, when they split these storied defined in the planning into tasks and calculate the amount of work needed to complete one. After that, they decide on how much they could take into this sprint. Then they define their goal. (Reynisdottir, 2013)

Sprint Goals should be defined as milestones of a higher-level product roadmap. At the beginning, they did not consider milestones at all but at later sprints, they had milestones, which corresponded with the product release plan. The length of this meeting was rather high at the beginning, but it was reduced to a two hour meeting with a break. Projects with high uncertainty can take longer, as more discussions are held. (Reynisdottir, 2013)

Work Breakdown: Breaking down work in a hardware project is rather hard. In this case, they also had their problems with breaking down. In the first release plan, they broke down their product into sub-tasks, in the latest ones the product was broken down into sub systems and components. At the beginning, they did not see the advantages of breaking down the product into smaller junks. Their first planning was a prototype of equipment which should have been placed in front of the main prototype machine, which would have been planned in the later release. They created one large twenty-point story at first, and were asked to split it down further. Then, they split it up into three parts: Designing a CAD model (13 points), making the design ready for construction (5 points), observe, and assisting during construction (3 points). They stated that the prototype had less functionality and was smaller than the main machine. They were asked whether they had been able to split it up as the main machine prototype. It was possible for them to divide it into four sub-system parts instead of sub-tasks. (Reynisdottir, 2013)

Sprint Review: According to Reynisdottir (2013) the mechanical team did the Sprint Review but was quite frustrated with that task. I could not demonstrate an entire product functionality, it was more like a meeting, were they talked about what the teams did, what their findings and hindrances were. They were not especially happy about showing unfinished CAD models or about talking of minor technical changes, since they did not think about getting some valuable feedback from the participants. The participants entailed two teams, the product owner, the in-house stakeholder and other interested people. Most attendees did not give any feedback and many people were from other embedded teams. Additionally, the teams felt quite uncomfortable with inviting the management into their meeting. The meeting plan was created by their Scrum aster in cooperation with the team, not the product owner who is envisaged for this, as stated in the chapter of Scrum 2.1. This meeting was more about presenting their status and receiving questions and feedback. In the theory, it is stated that this meeting's purpose is to inspect the products increment and adapt the product backlog, which did not happen in this case. The sprint review lengths should be two hours long for a two-week sprint, the review meetings lasted about one hour and thirty minutes per team. This was changed after creating the release plan. Now they lasted thirty minutes with fifty minutes for each team. This was done because of the frustration of the team with having so many meetings which are time consuming and with no valuable feedback. (Reynisdottir, 2013)

Sprint Retrospective: Retrospective was held regularly right from the beginning of the project. They had discussions on “what went well” and “what could have been better”. Some results were executed and some of them were not. In the first sprints this meeting helped them understand the Scrum framework better, but after some sprints, the team felt that the value of this meeting had been weakened. The length of this meeting was two hours long at the beginning and after some sprints it was reduced to thirty minutes, due to the same reason as in the Sprint Review meeting. (Reynisdottir, 2013)

Daily Stand Up: Daily Stand Up Meetings were held at the beginning of every day in front of their Scrum board. Sometimes they forgot that there was a meeting. They attended it only because of the Scrum master. Reynisdottir (2013) states that the team found this meeting useful for synchronizing their work, sometimes the Scrum master did not help the team with hindrances which were discovered in this meeting. (Reynisdottir, 2013)

The team did not explicitly answer the three questions defined in Scrum, but the reports answered these in some way. This meeting's purpose mainly is to report to the team not to the management. However, they did report to the Scrum master and to the product owner. This meeting's length was, as stated in Scrum 2.1, about 15 minutes long. (Reynisdottir, 2013)

Results of the Experiment

The project team could decide to continue using Scrum or not. They made the decision to continue using it. This was not obvious as, according to Reynisdottir (2013), the frustration concerning the framework usage increased. In most aspects, it was a success. They tried to use as much from the framework as possible. As a result, they stated that it is not possible to clarify whether Scrum is better than other methods for mechanical teams. However, it was useful for using it. Even if not the entire framework is implemented, the project still benefits from the following points:

- Increased cooperation and coordination
- Increased communication
- Better overview of the progress of the project
- Priority concerning work, less redundant work
- Knowledge and information is better distributed through the team and everyone else who attends the meetings
- More frequent feedback

(Reynisdottir, 2013)

Reynisdottir (2013) also states that it needs to be considered that adaptation have to be made because of the different nature of hardware or mechanical development. In this case, they made the following adaptations:

- The Team was not cross-functional.
- It is not always possible to produce a working product increment at each sprint, but the

focus should rather be on reaching a sprint goal (designing a part, or having a part ready for production).
- The release plan served as the product backlog and was not managed by the product owner.
- The Development Team created the backlog items and it was done in the sprint plan meeting.

(Reynisdottir, 2013)

4.2.4. Recommendation

- Top-Management support is crucial for success.
- Training the team and their manager to ensure that they know the theory behind the framework.
- An experienced Scrum master is needed for implementing the framework.
- Try to make adaptations to the framework if needed, to increase the team buy in of the members.
- Prepare for a reluctance to change.
- Work break down is hard at the beginning, and therefore needs some practice. The work should be only broken down into reasonable tasks, which help the team.
- Sprint Review and Retrospective helps with providing value for feedback and it ensures continuous improvement even if the team does not want it.

(Reynisdottir, 2013)

4.3. Summary

It was possible to implement an agile methodology into a hardware/mechanical environment in these Use Cases. However, due to the different nature of hardware and mechanical development adaptation had to be done in order to implement an agile methodology. The case of Huang et al. (2012) proves that it is possible to implement agility into a hardware and software environment without implementing a framework. Instead, taking only a few methods from it and using it is also an option. Additionally, from the study of Reynisdottir (2013) in the case of SAAB EDS, they did not implement Scrum but rather a Scrumish like method in a multidisciplinary hardware environment and in the last case of Marel GRB from Reynisdottir (2013), they implemented Scrum for developing a mechanical product with a mechanical hardware team and an embedded software team.

These cases should serve positive results for using an agile methodology or for just implementing some practices into it. The case of Huang et al. (2012) shows, that using a self-organized team, which had direct contact with the stakeholder, can lead to a quicker implementation of a changed

	John Hopkins	**SAAB EDS**	**Marel GRB**
Author	Huang et al. (2012)	Reynisdottir (2013)	Reynisdottir (2013)
Sector	Satellite development	Defense industry	Food processing
Agile method	only some practices	Scrumish method	Scrum
Challenges	new technology, limited budget, technical standards, flexible to changes	mindset change, work breakdown, hardware dependencies, changing priorities	involvement of development owner, hardware dependencies
Benefits	better communication, quicker response to changes, quicker decision making, more flexibility	better structured, quicker feedbacks through better planning	increased cooperation and coordination, improved communication, less redundant work, more frequent feedback
Takeaways	small teams, direct link to sponsors, co-located, test as early as possible, concepts as increments	find a good Scrum master, improve communication between Scrum master and the team, use Gantt charts for estimating delivery times	management support is crucial, train employees in the framework, experienced Scrum master, adapt framework to your needs

Table 3.: Summary of literature review for agile hardware development

request. Collocating the team here lead to more communication and provided more discussion, which reduced the number of meetings needed to present the results or to discuss changes. For discussions on the concept, a CAD model or simulation was projected and manipulated in real time. This helped them to have a more detailed view on the review team. Using a Scrum board and daily meetings helps to view the progress of the project and to adjust tasks with new priorities.

They were only in the ninth months in the SAAB EDS project, which entails 22 months. Here they tried to implement a Scrum like method and used almost every practice from Scrum. They used a “Next” column in their Scrum board to overcome idle time or waiting times. Breaking down the stories into tasks is rather hard at the beginning, but this improves after some time when the team gets some experience in it. They accomplished breaking down tasks from 10 days to tasks lasting 2-3 days. Priorities could be changed if there was a good reason for it. Mostly they stated that having a good Scrum master would lead the project to a success.

The case of Marel GRB provides a possible a way of implementing Scrum in a mechanical environment. The advantages of using it were that they had a better communication and cooperation of the team, redundant work decreased and there was generally more feedback. Adoptions were made to the framework in order to implement it. The team was not cross functional, because it was not possible for them to have all the knowledge needed in their team. It was also not always possible to have a working product increment at each sprint, but it was possible to set a reachable sprint goal. The release plan served as a product backlog, which also differs from the framework. The team itself created the item for the backlog. These adaptations were made with the help of the product owner, the Scrum master and the team itself.

Having two different teams with different fields for one product is quite problematic, as sprints need to be aligned and tested together. Alignment with the software team was done to accomplish testing. They also did not have a fixed sprint length but rather they tried to adapt the length depending on the circumstances.

Using agile methods in hardware development is possible according to these cases, but adoptions must be done to be able to implement the framework. Furthermore, it is not necessary to implement everything in this framework. Trying to implement one practice after another may improve the team's mindset change. It was not explicitly stated in all the cases how the testing was done. Only in the first case of Huang et al. (2012), incremental steps were done and tested to keep the focus on meaningful work. One possibility would be to create an early prototype using rapid prototyping, or testing just parts of a product instead of testing a whole prototype. The sprint lengths also depend on hardware development, as physical objects are involved in hardware, which usually take some time to be purchased, delivered and produced. Therefore, the sprint length should be adapted to one's needs. However, hardware development would benefit from using an agile method, as it also is a field of high uncertainty with rapid changes. Using an agile methodology would help solving these problems.

A summary in form of the table 3 was put together in order to have a better overview of the use cases and the takeaways.

Recommendations for adopting agile software development methods to hardware-, firmware development would be:

- Instead of implementing a framework, use some practices from agile development methods.
- Establish self-organizing teams with direct contact to stakeholder.
- Collocate the employees.
- A sprint increment can also be a CAD model or a simulation, rather than only a working prototype.
- Use a Scrum board with a next column.
- Use a release plan as the product backlog.
- Sprint length should not have a fix length.
- Having a good Scrum master is essential for changing the mindset of the employees.
- Define goals for each sprint.

Part II.

Practical Part

5. Agile Processes from AVL

The following chapter provides a description of the agile methods used at AVL. It will describe the agile software development method ALASKA, the agile hardware and firmware development method ALADIN and it will provide a comparison between the used methods and the literature.

The information for AVL's agile processes was taken from internal company documents. Therefore, the references will only be displayed as internal documents for ALASKA and internal documents for ALADIN.

5.1. ALASKA

ALASKA is a lean and agile software development process including a software portfolio management. It is designed to be scalable and holistic for more than 200 software developers. The ALASKA framework is build up in levels, and for each level, roles are defined which interact with each other (figure 9).

ALASKA is currently only used in a part of AVL namely the Instrumentation and Test Systems (ITS) department. The method is used for:

- Software products
- Specific extensions from customer to standard products
- Customer applications
- Software components

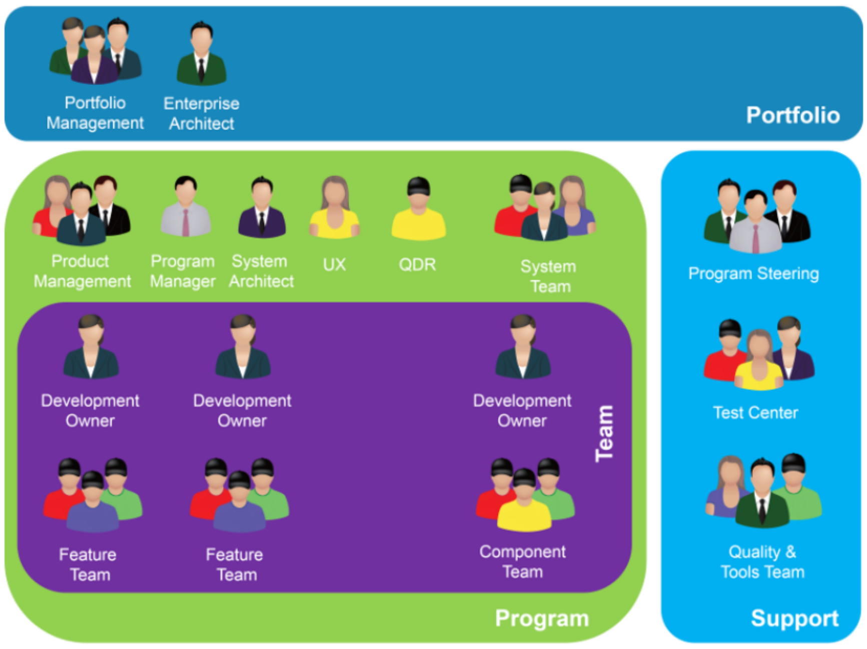


Figure 9.: ALASKA process (AVL, 2017)

5.1.1. ALASKA Process

The ALASKA development process is just a part of the AVL process structure for the “Research and development” process. ALASKA is a continuous software development process with a strict cadence of program iterations and team sprints.

The ALASKA process consists of three levels with different tasks: portfolio level, program level and team level (figure 10).

The portfolio level process is a continuous process that manages many software product portfolios. Its aim is to coordinate requirements concerning the portfolio and coordinating and supporting refinements of various programs (figure 10).

There are many different programs that are managed by the portfolio level process. These programs consist of product families. The program level process describes the roles and the activities within a program iteration. It consists of a team level process for many different teams and the program level process therefore synchronizes the activities (figure 10).

The team level process describes the roles and activities within a sprint. This process is based on Scrum and here the real development is realized.

The ALASKA process structure contains of

- 5 program iterations with a length of 10 weeks, each week encompasses:
  - 4 development sprints with a length of 2 weeks
  - 1 innovation and planning sprint with a length of 2 weeks

The software is tested and developed in teams. In order to coordinate these teams, all the activities are described in the team level process, which is performed repeatedly with a two-week sprint length. The produced software is then integrated into products and software systems.

Activities of finding product requirements, specifying requirements for the teams, and integrating and testing the software increments are described and coordinated in the program level process, which is performed during a ten weeks iterations length.

Requirements regarding the software product portfolio, which arise from various strategies from roadma, the coordination of requirements between programs and the communication regarding the development process are coordinated and described in the software portfolio management process. This process is performed continuously. It is regularly synchronized with the program level process if a new program iteration is to be prepared.

Software increments are delivered regularly in form of product releases and service releases. Different AVL processes plan these releases:

- PLC (Product Life Cycle)
- PIP (Product Innovation Process)
- Software Maintenance

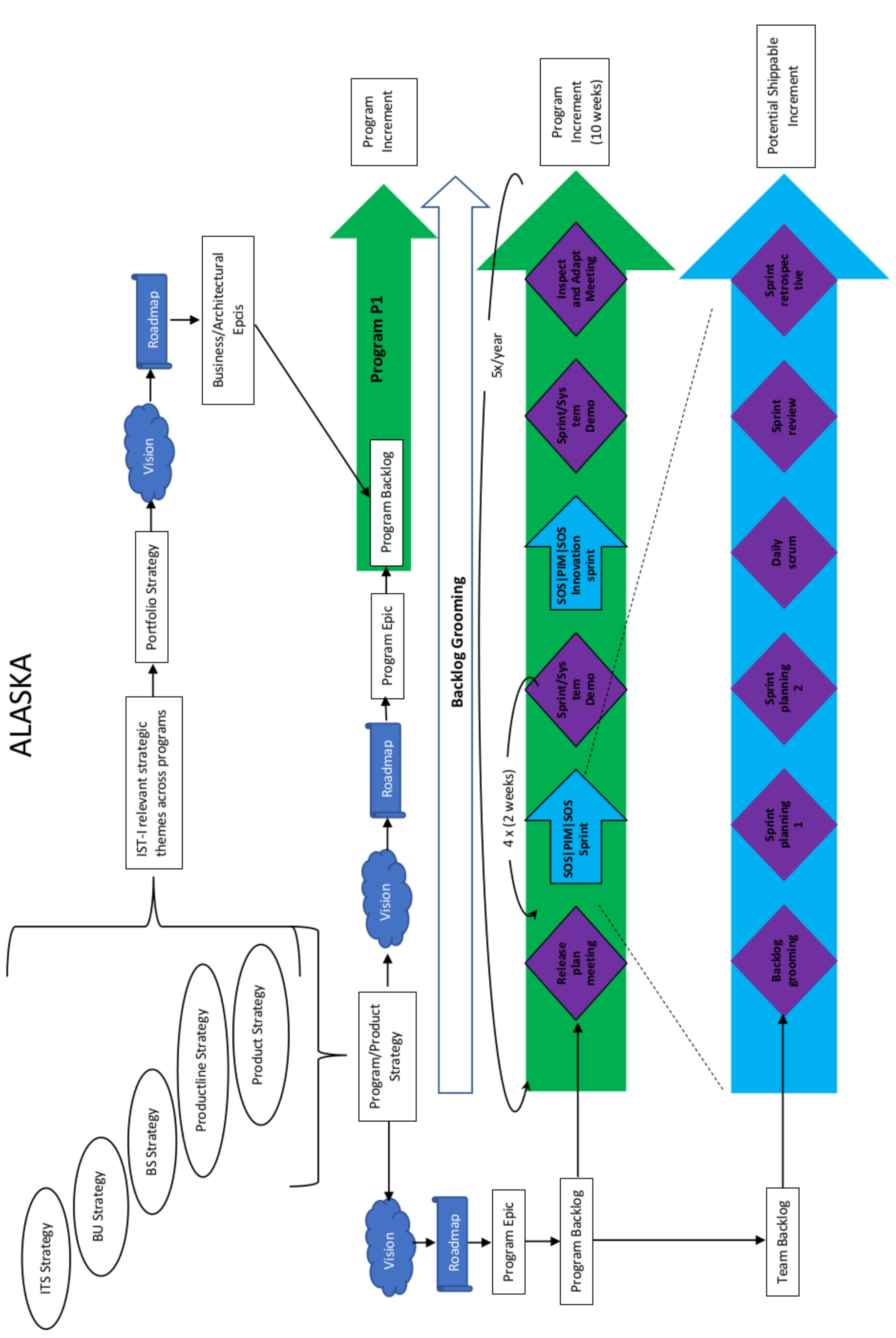


Figure 10.: ALASKA framework (personal design based on: AVL, 2017)

5.1.2. Team Level Process

The team level process describes the roles and the activities within a sprint. This process is based on the Scrum frameworks meetings, roles and events and it can be implemented independent from other processes of ALASKA (figure 11).

Activities in this process are:

- Software requirements refinement
- Software design, architectural and behavioral design
- Software build, implementation, unit test and software integration
- Software test

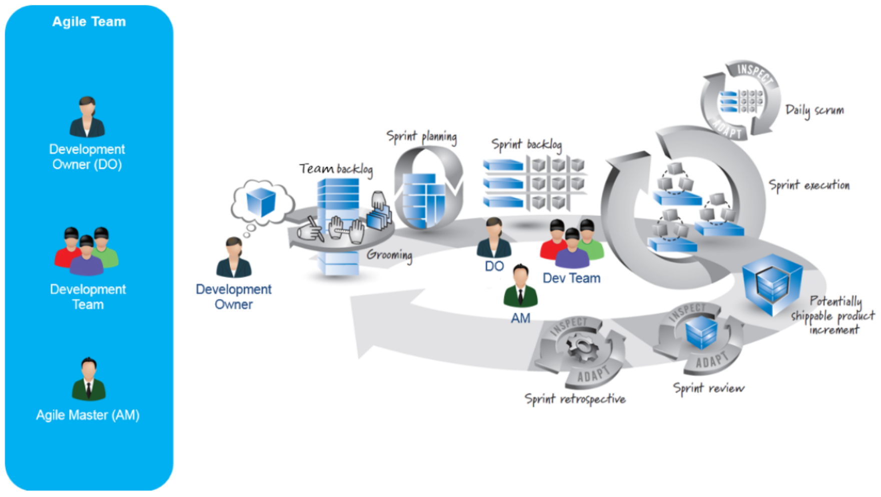


Figure 11.: ALASKA team level process (AVL, 2017)

5.1.2.1. Roles

Development Owner: Here the development owner represents the customer, and he or she is working with the product manager and other stakeholders (other development owner, the team) to define and to prioritize the team backlog. The development owner is the only team member who can add new stories to the system baseline. In ALASKA, the development owner has another task. He or she must participate in the product management team meetings and in planning and backlog/vision refinement session. The development owner shares the content authority with the product management; the responsibilities therefore have to be defined explicitly. The development owner is often more at the solution/technology/team facing side. His primary roles are:

- Backlog refinement: Taking input from the product manager and stakeholder. His responsibility is to build and maintain the team backlog.

- Sprint planning: He reviews and re-prioritizes the backlog for the sprint planning. This also includes the coordination of content dependencies with other development owners. He must accept the final sprint plan.
- Just in time elaboration: The process flowing must continue.
- Accepting stories into Baseline: Accepting user stories into the baseline and validating that the stories meet the acceptance criteria.
- Program Increment: The development owner has the responsibility of coordinating the content dependencies for each shippable increment by attending weekly development owner team meetings.
- Potential increment: As a member of the product management team, the product owner is involved in the preparation of the release planning and of the planning event.

Agile Team: The agile team consists of seven +/- two members including developers, testers, an agile master and the development owner. It should be a cross functional group that has the ability to elaborate, prioritize requirements and create a solution for them. They also write codes and tests for and against the solutions in a time boxed iteration. As in Scrum the team here is self-organized and self-managed. They are responsible for delivering the results that meet the stakeholder requirements. This includes estimating the amount of work, determining a technical solution, delivering a high quality product and continuously finding certain aspects that need to be improved.

The agile team is assigned to programs. Therefore, their team backlog is influenced by the large program backlog. To coordinate solution buildings, sprints of every team are aligned with the other teams in the program.

The agile teams in ALASKA are not solely based on Scrum but also on XP. Therefore, ALASKA provides some guidance on how to develop code, implement the practice agile architecture, continuous integration and test-first development.

Teams in ALASKA are collocated to communicate at all times and with all participates on several meetings. This includes the daily stand up, the sprint planning, the sprint demo and the retrospective. In some cases members, also participate in the system demo meeting.

Trusting others is a fundamental principle of ALASKA. This trust is established by common sprint goals, potential team increments objectives, regular feedback through loops and the inspect and adapt program.

There are two different types of teams in ALASKA: “Feature Teams” and “Component Teams”. Feature Teams develop features product and release orientated, while the Component Team develops components which can be used for several products. Therefore the developed components should be reusable and should represent the platform idea. They also support the feature teams by providing the desired components. Both teams work in programs. Component teams can change program affiliation only between sprints and can accept backlog items from other programs.

Agile Master: The agile master is a servant leader whose role it is to help the self-organized and self-managed team to accomplish their goals. He or she is responsible for reinforcing the rules of Scrum and ALASKA so that the team can understand and agree to them. Another responsibility of the servant leader is to facilitate the team progress and to keep the team focused on the goals of the iterations. It is also his or her duty to accomplish continuous improvements that includes helping the team taking responsibility for their actions and helping them to improve. The servant leader also has to eliminate hindrances that are beyond the team's authority and help to foster code quality practices.

In ALASKA, the agile master also has some additional responsibilities:

- Coordination with other agile masters, the system team and other stakeholders that participate in the release planning meetings.
- Helping with preparation and willingness for the release planning and the inspect and adapt ceremonies.
- Using standardized estimation tools to help estimate epics or features.
- Coordination of teams under architectural and portfolio control, system level integration and the regular system demos.
- Coordination with other agile masters in the Scrum of Scrum.

5.1.2.2. Artifacts

Team Backlog: is the collection of all the things a team has to work on for accomplishing their goal. It contains user stories, technical stories, defects, infrastructure work, spikes and refactors. The content of the backlog comes from different sources, including the program backlog, the team's local context and the needs of other stakeholders.

The program backlog consists of the features that should be delivered during a single program and of features for the upcoming programs. All the features which are planned to be implemented in a program iteration are then broken down into stories during the release planning and are then allocated to individual sprints in the team backlog. Estimation is usually done in story points.

For the team context additional to the stories that are needed for the program features, the team also has a backlog for refactors, defects, rehears spikes and other technical things. In addition, these things must be identified, estimated and prioritized.

Some of the backlog stories are in support of other teams and stakeholders objectives. These stories can reflect team dependencies, estimate spikes, research and other external commitments.

Story points measure the estimation of the backlog items (user stories, refactors and spikes). These story points are added together for higher requirement levels (features and epics). With these, the velocity of the team can be measured. While defects are estimated in working time units (ideal person hours and days), this part is not considered in the team's velocity. If a feature can be implemented by one team, then all the user stories are added together to estimate the feature. This estimation is then used by the product management for the Weighted Shortest Job First(WSJF) prioritization model. If the team can only implement a part of it, then the total estimation of the features is done including the roll-up from each involved team. Additionally, all features which consist of more of than one team are updated to estimate only the work which needs to be done. This helps to estimate how much work is remaining for the feature.

Because defects might also occur in the backlog, we need to consider how much time we can allocate for refactoring, bug fixing and maintenance. These things bring no immediate value compared to new user stories. For this, they are using **Capacity Allocation**. With this the team can estimate how much of their total effort can be applied for different types of activities. The allocation can be changed after each program increment.

Defects, local stories and refactors need to be estimated by using value/size or WSJF.

Sprint Backlog: contains a list of tasks that the agile team has to complete in order to create a deliverable increment of functionality. Sprint backlog items are estimated in hours and should not exceed the limit of one day, unlike the team backlog items. The agile team has the responsibility of keeping the sprint backlog up to date. This means that during a sprint, new tasks are added and adjustments are done. For better visualization of the sprint backlog, an agile task board is used. This helps to see how the progress of the current sprint is proceeding.

5.1.2.3. Events

Backlog Grooming: to prepare the team's backlog for the upcoming sprint plan meeting, backlog grooming is performed. In this meeting, the whole team including the development owner, the development team and the agile master has to be present. New stories and epics can be added, existing epics can be broken down into stories and the effort for existing stories can be estimated. This helps the sprint plan meeting to plan quicker without wasting time and most importantly, if the developers see the stories for the first time in the sprint plan meeting, errors in the estimation may arise.

Two different kind of estimations are used in ALASKA; the relative estimating and the estimating poker. The relative estimating uses the size (effort) of the smallest story relative to each story. Finished stories are then considered in the velocity of the team. With the velocity, it is possible to estimate how much work can be added to it in the upcoming sprints. It is also used to estimate how it will take to deliver larger features or epics.

A difference to normal Scrum is that for each team the story point estimated and thereof the velocity may vary. ALASKA tries to have a normalized story point velocity estimation, in order to estimate epics or features that require more than one team working on it correctly. Therefore, ALASKA demands that each story point means the same for different teams. This can be accomplished by converting story points to cost.

ALASKA uses the following algorithm to normalize the story points and velocity to a common baseline:

- Use of the pseudo Fibonacci sequence: 1,2,3,5,8,13,20,40,100(too big), ? (Unknown) for story points.
- No stories larger than 13 points are allowed in a sprint.
- At least 3 to 5 stories should be completed at each sprint.

Sprint Planning: is used to plan the work for the current sprint that can be accepted by the team. This helps the team to agree on the goals for the sprint and to have a commitment based on the team's capacity; consider complexity and size of each story and on depend stories and other teams.

The Sprint plan meeting refines the previous sprint plans from the release planning session. The team backlog for the sprint plan meeting is already pre-elaborated and aided by a separate backlog grooming session. Therefore the teams already have a backlog that consists of stories from the release plan meeting and of defects, spikes,refactors and stories that have been derived from the planning session. Feedback from different sources, from prior sprints (stories which did not meet the definition of Done) and feedback from system demos, the teams local demo and the program iteration objectives from the program iteration release planning are taken as an input for the sprint plan meeting. This meeting's duration is about four hours or less long.

For the meeting the development owner will have prepared some sprint goals based on the progress in the program iteration. Sprint goals and higher priority stories are reviewed. The teams discuss technical options on how to implement technical issues, nonfunctional requirements and dependencies and the team plans the iterations. The team makes estimations of how much effort it takes to complete a story and elaborations for the acceptance criteria. Based on the velocity of the previous sprint the team selects stories and breaks them down into tasks and estimates the tasks in hours to confirm that they have the skills and the capacity to complete these tasks.

The team determines in the beginning how much capacity each member has to perform in the upcoming sprint (availability, time off, potential duties). The capacity allocation is also considered. The average velocity of the last sprints constitutes a help to plan how much work can be done in the upcoming sprint.

After that, the team focuses on understanding and accepting one or more sprint goals, which are based on the team and program objectives from the release planning. Depending on different factors, sprint goals can be adjusted.

Based on the goals, the pre-refined backlog is reviewed. The team based on the difficulty discusses each story, size, complexity, technical challenges and acceptance criteria and the team agrees on the size estimation of a story. Stories (tasks) are written into a list until the whole capacity of the team is reached. After agreeing on the list, the list is transferred into the sprint backlog.

Usually the breakdown of the stories is done in a separate planning meeting. The best person to accomplish this task is than chosen, the task is estimated in hours, and other dependencies are considered to accomplish this task. Not all tasks have to be allocated to a person. However, it should be allocated to at least one person, as each member should work for some time. After that, each member can chose tasks on their own.

Daily Scrum Meeting: Each day the team members gather to answer three questions: "What have you done since yesterday?", "What are you planning to do today?", "Any impediments/stumbling blocks?" To do this each team member has to be prepared in order to answer these questions. The location should always be at the same place and the meeting should have a duration of 15 minutes length.

Sprint Demo: Is held when the team is able to demonstrate a functionality of the increment produced in the sprint. The sprint demo can be held at any time when there is something to show. Presenting it to the stakeholder helps to develop an understanding of how their requirements are implemented. Therefore, this may cause a good feedback. The feedback is an input for the sprint review meeting. While the sprint demo can be an independent meeting, it could also be a part of the sprint review itself.

Sprint Review: Is a meeting for inspecting the increment and adapting the team backlog. The meeting is held at the end of a sprint. The team reviews what has been done or what has not been done in the sprint and tries to improve the next thing to optimize value. In a four-hour time box, the team discusses what went well and what problems occurred during the sprint. Not only the increment is reviewed but also the timeline, budget, potential capabilities and the marketplace for the next release of the product are reviewed. The result of this meeting is a product backlog that defines the new items for the next sprint.

Sprint Retrospective: The purpose of this meeting is to reflect on the sprint, and find new ways of improving the process in a long term way. Each person who has any role in the iteration can participate. The meeting is held biweekly at a 60 minutes time box and is split into two parts, the quantitative and the qualitative. In the quantitative part, they assess whether they meet all the sprint goals. Metrics such as the velocity and other ones, which can be chosen independently, are analyzed. This should help to develop an understanding for their current process. In the quality part, they review the items that were identified in the last retrospective, and they analyze their current process to find some things they may improve in the next sprint. They ask themselves: “What went well’, “what didn’t go so well” and “what can we do better next time?” When these improvements are identified, larger improvements should be split up into smaller improvement items, in order to finish them quickly.

5.1.3. Program Level Process

The program level process describes the roles and the activities within a program iteration (figure 12). The program level process consists of the team level process and therefore synchronizes activities between the development teams. All the activities on this level are performed to create a program increment: Product requirements elicitation and refinement, product and system level integration, product and system tests, coordination of development activities regarding product release.

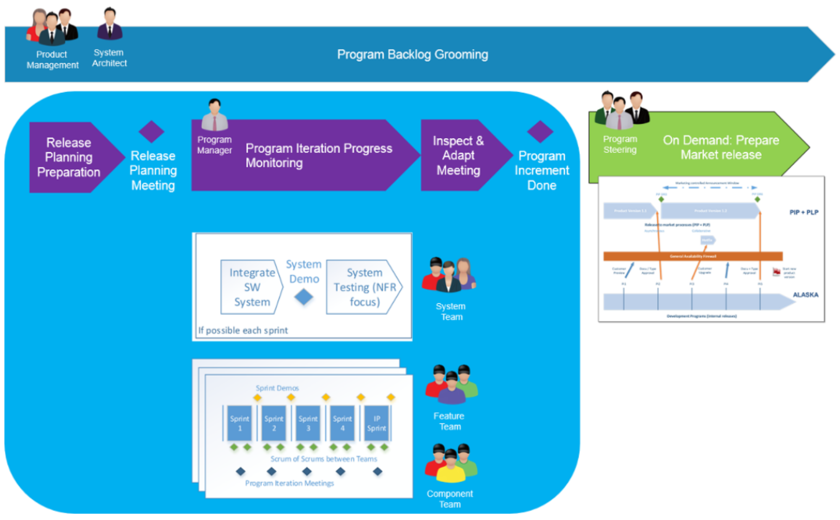


Figure 12.: ALASKA program level process (AVL, 2017)

5.1.3.1. Roles

Product Management: is responsible for defining and prioritizing the program backlog and for working with the development in order to optimize the feature delivery to the customers in balance with the AVL's technical and economic objectives. For projects that are more complex, this role is split up into two different Roles: Product Manager and Content Managers. The product manager is more responsible for the customer side. Product managers discover new markets for customers and they do market researches, create product strategies, plan and make decisions. The content manager's responsibilities are requirement-refining, communication with the agile teams, program backlog grooming, release planning readiness of program backlog and acceptance of feature delivery.

To summarize, the product management's responsibilities are:

- Working with the portfolio management to prioritize and determine business objectives. Therefore, the product manager must be a participant of the portfolio management team.
- Working with system architects in order to understand what the architectural epics are. While the product manager cannot decide which technology is used, he or she can assist on the decision-making.
- The product manager has to create a program vision, which contains the "what" question for each program solution. In order to do this, customer/market analysis has to be performed to know the customer's need. The complete vision is communicated to the team. Creating requirement documentation including use cases, scenarios, standards, helps the team understand what to do and how to break features into stories for the implementation.
- Managing the program backlog and program iteration release content.
- Participating in the release planning, release management and solution validation.
- Maintaining the product roadmap, at the end of each planning event.
- Building an effective product management/development owner team, to ensure that the development is effective and efficient.

Agile Program Manager: is the "chief Scrum master" whose responsibilities are facilitating program level processes and program execution, escalating impediments, managing risks and help driving continuous improvements on the program level. Additionally, agile program managers have the responsibility of facilitating the release-planning event, release management meetings, inspect and adapt workshop and the Scrum of Scrums.

System Architect: This role is not defined in Scrum. The system architect works with the agile team and focuses on design decisions to support future features. ALASKA defines two different kinds of system architects. Firstly, the Enterprise Architect, whose responsibilities lie on the portfolio level. He or she must assure that the It/Software strategies and technologies are aligned with the business needs of the enterprise. Secondly, the Software/System architect. His or her responsibilities are located on the program level. The system architect must have a high level of understanding the user vision and the system needs. System architects need to have an understanding of how to implement frameworks which are vitally necessary to support the user and business needs.

System Team: is a specialized team for the initial building of development infrastructures and program level activities, such as system level continuous integration and end to end testing.

Feature Team: is a long-lived, cross-functional and co-located team of seven +/- two members which complete end-to-end customer features one by one. All the knowledge needed for completing the features is in the team.

Component Team: is a team which focus lies more on the creation of one or more components of a larger product. The team creates assets or components which are then reused by other teams to create customer-valuable solutions.

Program Steering Team: is responsible for scheduling, managing and governing synchronized releases (program iterations) across one or more program or product lines. They coordinate and make it easier to deliver produced software solutions developed by the ALASKA program. The team typically consists of the Agile Program manager, senior representatives from product management, marketing, quality, development, program management and operations/deployment/distribution.

5.1.3.2. Events

Program Backlog Grooming: is almost the same as the backlog grooming for the team level process. Here the product management and the system architects meet up before the release planning to refine the backlog. This includes reviewing and updating features, establishing acceptance criteria, breaking down features into smaller task and determining the architectural features. For prioritizing the backlog, the WSJF method is used. Just before the release planning is held, the product manager makes the final backlog preparations, updates the vision, works with the development owner to socialize the backlog to the events and the system architect updates all his models and features and creates some use cases for illustration. In case of not having only new features or architectural features in the backlog, the "capacity allocation" method is used in the same way as in the team level process backlog grooming.

Release Planning Preparation: For the release, planning a preparation is held to ensure organizational readiness, content readiness and facility readiness. For the organizational readiness one has to consider that if the scope of planning is understood, which teams need to plan together, agreement on priorities among the business owner is established and if there are actual agile teams which have a development and test resource and know who the agile master and development owner are. For the content readiness one has to define the business context; a short briefing from the product manager including top ten features and architectures briefing to present some new features. For the facility readiness one has to ensure that there is enough space for all attendances and all the needed infrastructure has to be available. It is also necessary to have some communication channels ready.

Release Planning Meeting: is the seminal, cadence based synchronization point of the program. This meeting is an event in which all the participants have to be at one location and the event consists of all members of the program. The time box of this event is one and a half day at each program iteration. In case of different locations, this event may be held simultaneously with constant communication. It consists of a standardized agenda including the presentation of the vision, team-planning breakouts, commitment to the program iteration and release objectives for the next program iteration time box. The result of this event consists of the team program iteration objectives for each individual team, a summarized set of program iteration objectives (including release commitments), a program iteration plan for identifying the milestones during this period, and a vote of commitment for the objectives from the program. After the planning, the roadmap is updated so that it reflects the results and the forecast for the future program iterations.

Program Iteration progress monitoring: For monitoring the progress, this is the best option. It consists of bi weekly reports of the program steering team. The purpose is to measure the progress of the used software. Other tools for measuring the progress are the release burn down chart and the feature status report. Additionally, for programs consisting of more than one team, a Scrum of Scrums is held bi-weekly to monitor the progress of each team.

Inspect and Adapt Meeting: is held after each program iteration. It is the same as the sprint review and the retrospective in the team level process. Here it is possible to reflect upon problems and to solve them, to make improvement actions and to increase velocity, quality and reliability for the next program iteration. This is done by showing the current state to the stakeholder. With the feedback from the stakeholder and the collected information, a retrospective and a problem solving workshop is done.

Program Increment done: it is used to check whether the current program increment fulfills the definition of "done" for the program iteration and the potentially shippable increment. The requirements are evaluated in the inspect and adapt meeting. Based on that, the decision is made to release the program increment on the market.

Program level integration and Testing: creating the solution is mostly done by the agile design, build and test teams. However, for integrating and evaluating of a full system solution, an extra team is built on the program level, namely; the system team. The system team then integrates the software increment into a product or a software solution in order to ensure that the software runs stable. After that, they prepare the system for a system demo to demonstrate the functional capabilities. After a successful system demo, the system is tested with focus on nonfunctional requirements. This activity can be done during each sprint at the sprint demos. Another approach would be to do this for individual “done” stories/features.

Program iteration: consists of a certain structure and sequences for the work of the agile development teams. One program iteration produces a program increment release, within ten weeks. It consists of four development sprints and one innovation and planning sprint. This is the same for all the different agile teams within a program. The starting point and the finishing point are aligned in a way that the dates are as close as possible.

Market Release preparation: shows how and when to release a product increment. Therefore, a cooperation between the ALASKA process and the product innovation process (PIP) and the product launch process is done. It consists of releasing program increment boundaries after everything is finished. Another way would be to release more frequently or less frequently depending on the given needs. A more realistic way is to have a combination of all three methods: Release whatever you want whenever it makes sense.

5.1.3.3. Artifacts

Program Level Backlog: Consists of all the upcoming work necessary to achieve the program solution. The difference between the program backlog and the team backlog is that in the program backlog architectural features are also inserted. The responsible person for maintaining the backlog is the product manager. For having a good balance between feature development and architectural features, the system architect and the program management decide it with the help of capacity allocation. Prioritizing and re-prioritizing is done using the WSJF method and the planning boundaries.

Architectural Features: are technical systems that allow the developer to implement business features that deliver solution values. Identifying, splitting these features into tasks and prioritizing is the responsibility of the system architect; implementing is done by the agile teams.

Architectural Runway: is the ability to implement high priority features in a near term program increment without having an excessive delay inducing or even redesigning. Architectural epics are split into smaller features. Each feature should be implemented within a program iteration. Due to this, the implementation and testing are behind the scenes until the capabilities exist to support the implementation of new business epics and the program level features.

System Demo: to measure the progress of working software, ALASKA implements two different kind of demos at the end of each sprint: the sprint demo (for individual teams) and the system demo (for program level). The system demo provides an integrated perspective on all the new software that is created by the teams in the program in the recent sprint. With this demo it is possible to gain system level and customer feedback.

5.1.4. Portfolio Level Process

In ALASKA, the portfolio management process describes the roles and activities for managing the software product portfolios (figure 13). The difference between the team and the program level is that this process is a continuous one. The aim of this process is to coordinate requirements concerning the product portfolio, coordinating and supporting the refinement of cross program requirements and managing synchronized hand overs of cross program requirements to the ALASKA development program. Moreover, this process seeks to monitor and communicate the progress of the implementation of cross program requirements and to coordinate creation, maintenance and communication of the portfolio release roadmap, which is based on the roadmaps provided by the product management, development and the customer projects.

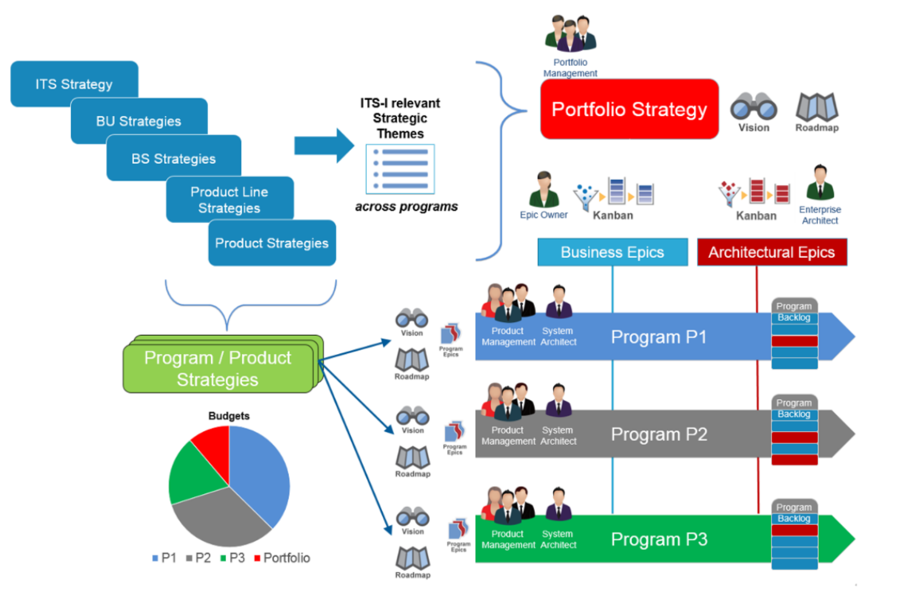


Figure 13.: ALASKA portfolio level process (AVL, 2017)

5.1.4.1. Roles

Portfolio Management: is the highest role in ALASKA. It represents the investment, return and content authority. The portfolio management consists of business managers, and executives who have the responsibility of strategy and investment funding, program management and governance. In order to define and implement the portfolio product/solution strategy, a profound understanding of the enterprise business strategy, technology and financial constraints is needed. This task needs assistance from the project or program management office that shares the responsibilities of guiding the program execution and governance.

Epic Owner: this role is presented in ALASKA because sometimes epics have cross cutting, crossing programs, and business units. This role has to accompany the epic through the Kanban system and developing business cases. After approving it working directly with the key stakeholder at the affected programs.

Enterprise Architect: works with business stakeholders and system architects in order to implement technology across many programs. This role has the responsibility of choosing a technology that supports the current investment themes, assuring that individual program and product strategies align with the enterprise objectives, developing and deploying infrastructure, implementing them incremental and having feedback from the teams.

5.1.4.2. Artifacts

Portfolio Strategy: has to consider many different strategies from the strategy of the business unit to the strategy of the customer. Each of these strategies have an impact on the portfolio and from them two relevant for strategies are derived for ALASKA: Strategic themes that are the input for the portfolio strategy and which contain business requirements. They cover larger cross program initiatives and the product line strategy that is created by the segments who develop products based on the related business segments. This is an input for the product management and their product strategy and in ALASKA, this is also the basis for the program strategy.

Portfolio Vision: represents the elaboration and commitment of action to the enterprise business strategy. It holds the programs, budgets and portfolio backlog epics that represent the vision of a portfolio of programs. The budget for the programs are allocated per program. This provides each program with the authority of making their own decision. Other elements for the portfolio are managed through the portfolio backlog. For this a special budget is reserved, namely the portfolio backlog budget. It contains the budget that is reserved for cross cutting business and architectural epics.

Portfolio Roadmap: consists of a mid-term timeline of proposed product releases and their content. The timeline is based on the vision that is based on the portfolio strategy. It helps to order the strategy themes based on the expected market availability. It is used as an input for ordering the portfolio and program backlog before having a release-planning meeting of the programs.

Portfolio Epics Kanban: consist of a Kanban system for business and architectural portfolio epics. It is used to have more visibility on strategic business initiatives, establishing a structure and making the process more visible. It also provides work in progress limits, helps to drive collaboration among the stakeholder, architecture and the development team.

Business Epics: are large cross cutting customers facing initiatives that need new development in order to realize a certain business benefit. Approved business epics are managed in the portfolio backlog. They are processed through many states of maturity until they are finally moved to implementation. This process assumes economic analysis of the business driver behind the epic, its technical impact and strategies for incremental implementation. A business epic implementation can take many program iteration and product releases and may affect multiple release trains, applications, solutions and business platforms. Additionally, it also affects multiple departments, business units and other end-to-end business value.

Architectural Epic: are large cross cutting technology initiatives that are necessary to evolve portfolio solutions that support current and features business needs. Architectural epics are approved and then managed in the portfolio backlog. As for business epic, also the architectural epic will pass through many states of maturity until it is moved further to implementation. The process provided also assumes an economic analysis of the business and technology driver, its technical impact and the strategy for implementing it incremental.

Strategic Themes: are topics of strategies that are relevant for the development programs in ALASKA. They describe larger cross program initiatives. The themes have to be defined into business requirement stories in order to be put into the portfolio Kanban.

Business Requirement Stories: is the collection of information about requirements that internal stakeholders have concerning the product portfolio. They are the input for the portfolio epic Kanban system in the workflow of the Kanban system. Additional information is collected to support the decision-making, such as market driver, customer needs and benefits, competitive advantages, technical feasibility, financial reward estimation, etc.

Program strategy: is the strategy for development of customer values. In a technology and development focused way, it defines how the market requirements are fulfilled so that it is resource efficient, while it also considers the whole product life cycle and the total cost of ownership. The program strategy consists of a product strategy for products, synergies between product and resulting software solutions, in development and common component and platform strategies. The strategy is developed by the product management with a focus on the content manager, system architect, business owner and line managers with the help of the portfolio management and enterprise architect.

Product strategy: is rather market focused and contains all the information about market access, business development, information about competitors and their strategies and vision, a roadmap for the marketing and development of the AVL's products.

Program/Product vision: describes the solution that should be developed. It contains the features that meet the stakeholder's needs, a market description and customer segments. It main content is a set of features (functional and nonfunctional), which is the input for the program and represents the boundaries in which decision for new user stories are made.

Program and product roadmap: represent a three to six month timeline of program iterations and release milestones. It contains three different visibilities: high-confidence for the next program iteration, medium for the one after and low confidence for longer term (due to a change of business objectives). The roadmap is developed and maintained by the product management.

Program epics: are epics for a single program train. They can have the size of many program iterations. They can be identified by the portfolio backlog or may occur locally. They always require an analysis and impact assessment.

5.2. ALADIN

ALADIN (AVL's Lean and Agile Device Innovation Framework) is an agile development method based on Scrum for the entire hardware development within the ITS department. The ALADIN process is not finished and is currently still in the test phase. In this phase, the process is being changed and updated to fulfill the vision of agile in a hardware environment.

5.2.1. ALADIN Process

As ALASKA's team level, the ALADIN process is based on the Scrum framework. ALADIN consists of the same meetings and roles with some additional roles and events (figure14).

5.2.2. Roles

In the ALADIN framework, there are four major roles: Development team, Scrum master, product owner and the product manager. The Scrum team is called "core team" in ALADIN it consists of the Scrum master, the development team and the product owner. The team itself is self-organized and cross functional. It has all the knowledge needed to fulfill the requirements without depending on external ones. The team choose itself how to work and who should do what.

Moreover, ALADIN has an extended team that consists of the product manager and other stakeholders such as manufacturing, procurement and service.

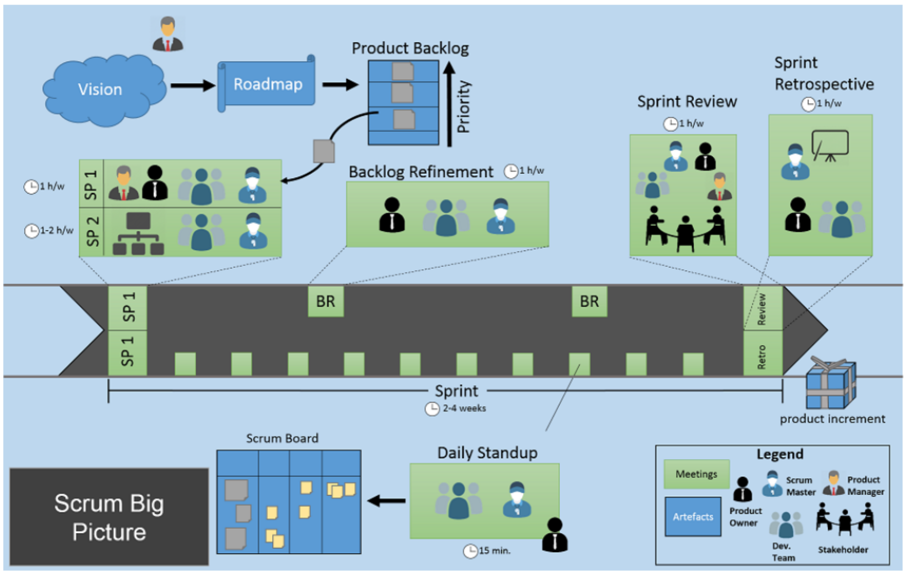


Figure 14.: ALADIN framework (AVL, 2018)

Product owner: has the responsibility of maximizing the value of the product and of maximizing the efficiency of the development team. The product owner is also responsible for managing the product backlog, which includes expressing product backlog items, ordering the items, ensuring that the product backlog is visible and transparent to all and ensuring that the development team understands the product backlog items. Only the product owner can change or add new items to the product backlog. This ensures that nothing will disturb the development teams' work.

Additionally, the product owner has to align his work with the product manager who is responsible for the ITS product innovation process, the costs and scheduled plans.

Scrum Master: has the responsibility of ensuring that Scrum is understood, which means that all the rules are followed, and everyone knows the theory and practice. The Scrum master is a servant-leader of the Scrum team, he or she tries to minimize the unwanted interaction with the Scrum team and helps people outside from the Scrum team understand which interactions are useful for the Scrum team. Therefore, the Scrum master has different purposes concerning the product owner, development team and the organization itself. Concerning the product owner the Scrum master helps finding techniques for managing the product backlog, ensuring that the product owner knows how to arrange the backlog to maximize the value, understanding agility and facilitation scrum events when needed. The Scrum master helps the development team by coaching the team to be self-organized and cross-functional, creating high value products and removing impediments. In terms of the organization, he or she helps by leading and coaching the organization concerning the adoption of Scrum, planning Scrum implementations, causing changes which increase the productivity of the scrum team and helping by working with other Scrum masters to increase the effectivity of Scrum in the organization.

Development team: works to create a potential releasable increment of a product at the end of each sprint. However, in ALADIN the main goal is to have feedback about internal deliverables rather than realizing a product increment for the customer. The development team is a self-organizing and cross functional team. Nobody tells them how they should do their work and they have the needed knowledge to solve the backlog items. The accountability should belong to the whole development team and not to individual members. The development teams' size should be big enough to complete significant work within a sprint but small enough to remain nimble.

Product manager: has a more market/customer based view. Her or his role is the same as in the product innovation process of AVL. Product managers needs to work closely with the product owner. They attend Scrum meetings such as sprint planning and review. The product manager represents the customer and he or she is responsible for contributing to the backlog definition and refinement. Due to the market or customer based view the product manager is responsible for the product strategy, the product vision, the product release roadmap and the creation of market based requirements. Regarding the product innovation, it is the product manager's responsibility to create business cases, definitions of target goal production costs and to review the Return of invest.

5.2.3. Events

ALADIN entails the same events that the Scrum framework does. All of these events are time boxed and everyone can attend them, except the retrospective.

Daily Standup: is held daily with only fifteen minutes with the development team, the Scrum master and the product owner if necessary. In this meeting, the development team will explain what they did yesterday, what they will do today and whether they see any impediments. These meetings help the development team to understand what everyone is doing, what they are going to do in the next 24 hours and how they can work together to reach their goal. The development team will use this meeting to inspect the progress towards the sprint goal and the progress towards completing the work in the sprint backlog. The Scrum master is responsible for ensuring that all the development team members will participate in the daily Scrum. Additionally, it must be ensured that the meeting will not exceed the limit of fifteen minutes. Daily Scrum improves the communication, removes unnecessary meetings, helps to identify impediments and improves decision making and the knowledge of the team.

Sprint planning 1: is held once at the beginning of the sprint with a two-hour time limit for a two-week iteration and four hours for a one-month sprint. The development team, the Scrum master, the product owner, the product manager and other stakeholders get together to discuss what could be derived from the resulting increment of the upcoming sprint and what work is needed to achieve the delivery of the increment. To carry out these tasks, the development team will forecast what functionalities will be developed in the sprint, while the product owner discusses the objective of the sprint and what product backlog items have to be finished in order to achieve the sprint goal. As an input for this meeting, the product backlog, the latest product increment, the capacity of the development team and the past performance is needed.

Sprint planning 2: is held once at the beginning of the sprint, with a two-hour time limit for a two-week iteration and four hours for a one-month sprint. In this meeting the development team, the Scrum master and product are the participants for this meeting. Here the development team will decide how they will implement functionality into a “Done” product increment during the sprint. The selected product backlog items for this sprint are inserted into the sprint backlog. The items the development team selects vary in size and estimated effort. These items will be decomposed to tasks within one day or even less. The product owner helps the development team to clarify the selected product backlog items and to make some trade-offs. If the development team has too much or not enough work, they can renegotiate the selected product backlog items with the product owner.

Backlog Refinement: is held once or twice a week with a one-and-a-half-hour time limit for a two-week iteration. In this meeting the development team, the Scrum master and the product owner get together to add some detail and they estimate and order the product backlog items. The Scrum team decides when and how the backlog refinement is held, but even then, the product owner has the power to update the product backlog items at any time. The product backlog items that can be implemented in one sprint are deemed as “Ready” for the selection into the sprint backlog.

Sprint review: is held once at the end of the sprint with a two hours’ time limit for a two-week iteration and four hours for a one-month sprint iteration. In this meeting the development team, the Scrum master, the product owner, the product manager and other stakeholders get together to discuss what has been done. They seek to receive feedback by demonstrating what has been done, discussing what went well and what problems occurred during the sprint. They discuss the product backlog and adapt it if needed. They also discuss what they want to do in the next sprint, analyze the marketplace or the potential of the product change and review the timeline, budget and marketplace for the next release of the product.

Sprint retrospective: is held once at the end of the sprint with a one-and-a-half-hour time limit for a two-week iteration and three hours for a one-month sprint iteration. In this meeting the development team, the Scrum master and the product owner gets together to inspect the last sprint regarding the participants, tools, and processes. They identify and order major items that went well and potential improvements and create a plan for implementing the improvements. This helps to improve the team within Scrum and makes it more effective and more enjoyable. While improvements are identified in this meeting, they can be implemented at any time.

5.2.4. Artefacts

ALADIN consists of some Scrum and some additional artefacts.

Product backlog: is an ordered list of everything the product needs and what should be changed. It contains a list of all features, functions, requirements, enhancements and fixes for the product in feature releases. Each of the product items has a description, an order, an estimation and a value. One product backlog can also be used for multiple Scrum teams in order to have a clearer view on the product's implementation progress. The adding of details, estimations and an order to the product backlog items is done in the backlog refinement meeting with the help of the product owner and the development team. However, only the development team has the final choice of how much an item is estimated. The product backlog can be used to determine how much work is remaining for reaching a certain goal. At every time the product owner has the possibility of tracking this progress, but he or she must do it at least at every sprint review. Various other tools help to monitor the progress such as a burn-down, burn-up or cumulative flowcharts.

Product vision: is developed by the product manager. It shows the goal of the product and provides an understanding of the requirements and the content. The product vision is part of the product strategy and must therefore be easy to communicate to other people. Product manager and product owner must ensure that the product vision is always transparent and visible for the whole Scrum team.

Scrum board: is the visualization of the sprint backlog. The sprint backlog is a set of selected product backlog items for a sprint; a plan for delivering the product increment and realizing the sprint goal. The development team forecasts which functionality will be in the next increment and what work is needed to solve it, and in the sprint backlog, this is made visible. The sprint backlog is modified throughout the sprint; new work is also added to the sprint backlog by the development team. Only the development team can change the sprint backlog. Due to the sprint backlog, it is possible to track the remaining work for achieving the sprint goal. This is done at least at every daily Scrum.

Product increment: is the sum of all the completed product backlog items in the current and all the previous sprints. At the end of each sprint a functional increment must be completed, which means it must fulfill the definition of "Done" and must be usable.

Roadmap: is the responsibility of the product manager. It shows what kind of products should be available, and what the main product is and what kind of options exist for it. The roadmap is unified with the roadmap of each business segment, and has to be made visible by the product owner to ensure transparency. For the planning stage, the rules of Scrum need to be considered.

Artifact Transparency: in order to make precious decisions, Scrum must be transparent. This means that all the decisions to optimize value and to control risks are based on the previous artifacts and if these artifacts are not transparent decisions may be flawed. To establish transparency, the Scrum master must work with the product owner, the development team and other stakeholders in order to understand that the artifacts are transparent. If the artifacts are not transparent the Scrum master has to steer and try to increase the transparency with the help of the Scrum team and the organization using tools, learning, convincing and changing things.

Definition of "Done": varies greatly among different Scrum teams. To ensure transparency, everyone has to understand what "Done" means. A simple definition could be that every increment is additive to all the previous increments and is thoroughly tested, ensuring that all the increments work together without any problems. The definition of "Done" can be extended with more criteria to ensure higher quality.

5.3. Comparison ALASKA and ALADIN

This section compares the software development framework ALASKA with the hard-/firmware development framework ALADIN (figure 15). It will show the differences and similarities between these two processes. The lightning icons shows were aligning and synchronizing problems might occur, for example the frameworks sprint length must be synchronized in order to produce an increment at the same time.

ALASKA is the agile software development framework of AVL's ITS department. The ALASKA process is already released and used in software development projects in the company. It describes a scalable agile way to develop software development for more than 200 software developers. ALADIN is the agile hardware/firmware development framework in the ITS department. It is currently under development and therefore not finished.

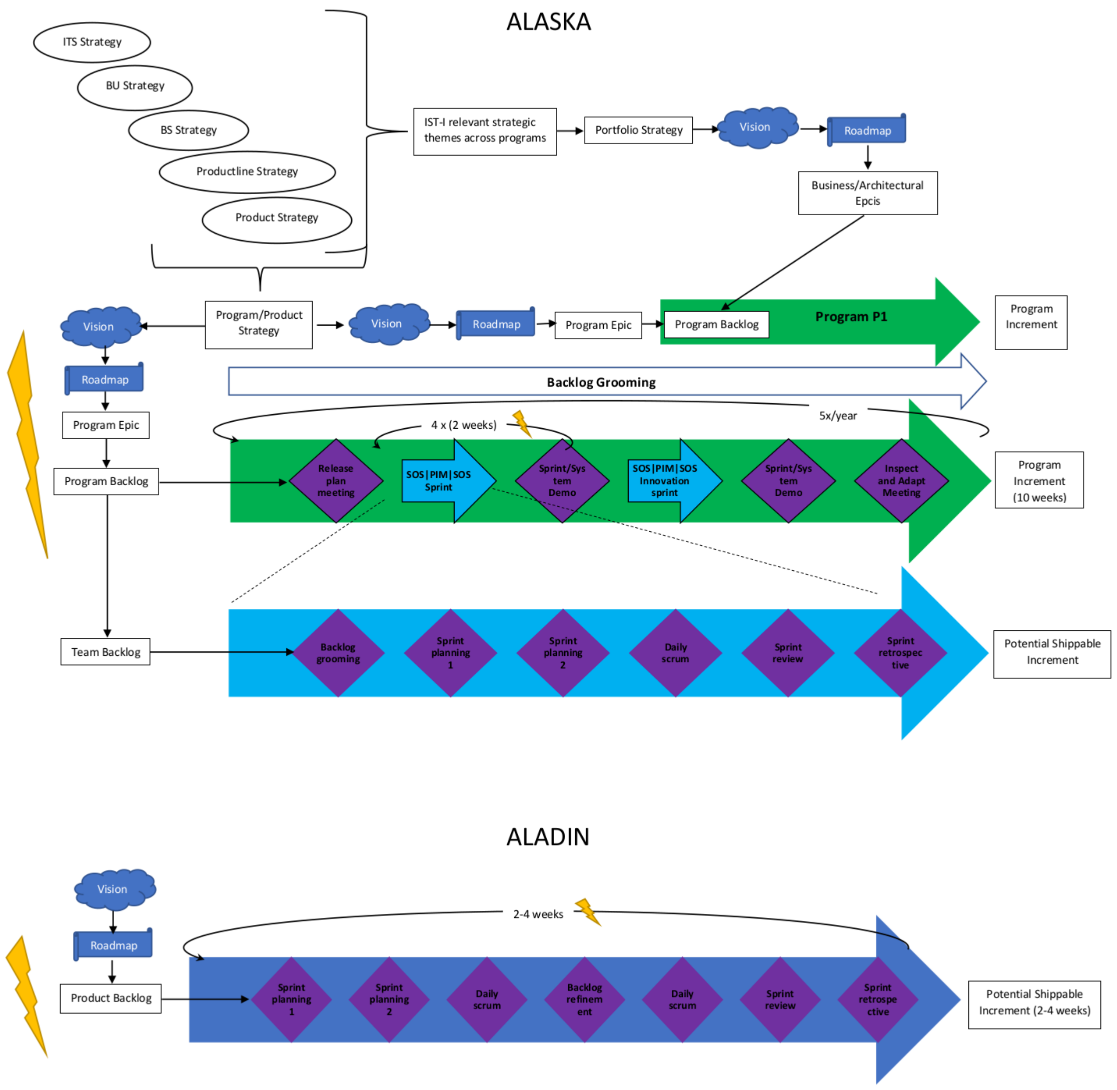


Figure 15.: ALASKA and ALADIN comparison (personal design based on: AVL, 2017, 2018)

5.3.1. Process

The ALASKA process is much bigger than the ALADIN process, as ALASKA consists of three different levels: Portfolio level, program level and team level. In each level different work is performed. In the portfolio level the strategies of the ITS department, business unit strategy, business segment strategy, product strategy and product strategy are taken into account to create a portfolio strategy and a program/product strategy, which is then the input for the vision and the roadmap of the business and for the many programs and their program/portfolio backlog. While ALADIN has not a real alignment with the strategies of the departments/ segments, the input for the program backlog derives from the vision of the product manager for his product and from the demand of the market.

ALASKA consists of many different programs. Each of them has an own program backlog but the same portfolio backlog, which consists of business/architectural epics that are important for every program. In the program level the planning for the next product release and steering/monitoring the development is done. An iteration of a program consists of ten weeks. In these weeks, a release plan meeting is done at the beginning to plan what actions need to be taken in order to create a program increment. These requirements are then distributed among the teams in order to create an increment. The program level consists of different teams that all have four sprints and one innovation sprint. During these sprints the deliverables are developed. To steer and control the development of these teams, a Scrum of Scrum meeting is held every week and a program iteration meeting is held biweekly. This helps to monitor the current development progress of each team. At each sprint a sprint demo and a system demo are done. The system demo checks if all the different components of the team's deliverables are working together without any problems. At the end of the ten weeks an "Inspect and Adapt" meeting is held, which can be compared with the sprint review/ sprint retrospective. This meeting's purpose is to gain feedback from the stakeholder and to solve problems that occurred during developing the program increment. After this meeting, a program increment is done which could have been already released into the market as a first version. While ALADIN has no unique program process, the only similarity to this is that it contains a product backlog, which is the same as a program backlog. It includes all the requirements of a product. Moreover, ALADIN does not take considerations regarding how different teams should work together; a Scrum of Scrum meeting is missing. This is a hindrance for the development of bigger products with more teams.

There are minor differences in the team level process between ALASKA and ALADIN. Here ALASKA chooses a fixed two weeks sprint, which starts with a backlog grooming that takes five to ten percent of the time of a sprint. After these two, the two sprint planning meetings are held one after another, and the developing is monitored through a daily Scrum. At the end of a sprint, a sprint review/demo and a sprint retrospective are held to check whether the deliverable is correct. While ALADIN has a two to four-week sprint, which also starts with two sprint planning and a daily Scrum afterwards, a backlog refinement is done twice (if necessary, else just once) within a one hour per week time box. Additionally, at the end of a sprint review/demo and a sprint retrospective is held to check whether the deliverable is working and to receive some feedback.

5.3.2. Events

On the team level, ALASKA and ALADIN have almost the same events; the only difference is that ALASKA starts with the backlog grooming.

Backlog grooming/ Backlog refinement: To prepare the backlog for the upcoming sprint planning, it takes about five to ten percent of the sprint time, whereas in ALADIN the backlog refinement meeting is held once or twice a week with a duration of one hour for a one-week sprint. In ALADIN, the backlog refinement adds some detail and estimates and orders the product backlog, whereas in ALASKA the backlog grooming entails adding new epics and stories, extracting stories from epics and estimating the effort for existing stories. They prefer to have the meeting at the beginning in order to have a preliminary planning before the sprint planning to ensure that the sprint plan meeting will not take too much time.

Sprint planning: In both ALASKA and ALADIN the sprint planning meeting is split up into two different meetings. For the first planning meeting both are discussed. It is discussed what can be done in this sprint and they make a forecast on a selection of the items for this sprint. After that, they create a sprint goal and every team member must be committed to it. This meeting lasts about two hours for a sprint of two weeks in ALADIN. In the second sprint planning meeting in ALASKA and in ALADIN the stories are broken down into tasks with a maximum size of one day and individual team members can take over tasks. The second meeting also lasts two hours for a two-week sprint in ALADIN. For ALASKA, the timebox is limited to four hours for a two-week sprint. For both sprint-planning meetings, it can be split up depending on the necessity.

Daily Scrum/Daily Standup: ALADIN and ALASKA are similar in this aspect. In both of them, framework meetings discuss on what the team members did yesterday, what they plan to do today and what kind of impediments they experienced. However, with ALADIN these questions are more sprint goal focused than in ALASKA, such as “what did you do today to help achieving the sprint goals”.

Sprint Review/Demo: ALADIN and ALASKA both have a sprint review/demo. In this meeting, they gain feedback by presenting the stakeholder what they have “Done” and what has not been “Done”. They discuss what went well in the sprint and were problems occurred and how they were solved. Additionally, discussions regarding the backlog and the market are held to determine whether the potential might have changed and what they have to do for the next sprint. However, only in ALASKA there is also a sprint demo, where a functionality of a software increment is presented to give the stakeholder a chance to learn how their requirements are implemented into a working software. This can be done at any time when there is something to demonstrate.

Sprint retrospective: ALADIN and ALASKA both have a sprint retrospective at the end of each sprint. In this sprint they both try to inspect the following things: how did the last sprint went, what went well, did they meet their goals, were there potential improvements and how can these improvements be implemented in the next sprint. In ALASKA, they split up the meeting into two parts; the quality and the quantitative review. In the quantitative review, they gather all the metrics they decided to analyze. These metrics are made visible for the whole team to have a profound understanding of their process. While they try to find one or two things that they can do better in the next sprint on the qualitative review, some improvements which would take longer to implement can be divided into smaller improvements. The retrospective meeting will last one and a half hour for a two weeks sprint in ALADIN and one hour for a two-week sprint in ALASKA.

5.3.3. Roles

ALASKA defines three different roles on the team level: the development owner, the development team and the agile master. In ALADIN, there are five roles: product owner, Scrum master, development team, product manager and further roles/stakeholder.

Development owner/product management/product owner: in ALASKA and in ALADIN these roles have the content authority for the solution. Both of these frameworks try to divide the labor into a more market-orientated view for the product management. His responsibilities are having a market/customer view and creating and maintaining the product strategy (vision, roadmap).

The difference between ALASKA and ALADIN is that there is no product management in the team level of ALASKA but rather on the program level. Depending on how complex a product is, ALASKA divides the product management into a product manager which is more customer oriented into and a content manager which is more content/development oriented. The development owner or the product owner have both a more development-oriented view. They both own the backlog, communicate directly with the development team, order the backlog items, establish story acceptance criteria and work closely with the product management. Moreover, the development owner in ALASKA participates in an extended product manager/development owner team, where they contribute to the vision and to the roadmap, actively work in the program level in refining the program backlog and on the preparation for the release plan.

Differences to ALADIN concern not only the product owner but also the responsibilities for compliance with the ITS product innovation process. The product owner also provides and maintains the cost and schedule plans.

Scrum master/ Agile master: in both ALASKA and in ALADIN the Scrum master/agile master is a servant leader whose responsibilities are helping the team to follow the Scrum rules, helping them meet their daily iteration objectives, removing impediments and facilitate continuous improvements.

Difference are that in ALASKA the agile master must coordinate with other agile masters in a Scrum of Scrum in the same program, with the system team and other stakeholders in the release plan meeting. Additionally he or she must coordinate the team's effort, the system level integration and regular system demos under the architectural and portfolio governance.

Development team: in ALASKA and in ALADIN the development team entails about five -/+ two members, not including the agile master/Scrum master and the product owner/development owner. In both the team is responsible for self-management, estimating the size and the work of product backlog items, finding technical solutions for the tasks, delivering the product, establishing an adequate product quality and finding new ways to improve. This means that the team has to be cross-functional.

A difference here is that in ALASKA the development teams are assigned to programs. They are cooperating with other teams to develop more valuable increments of working system-level software and their backlog is mainly driven from the program backlog. Additionally different development teams exist in ALASKA: component teams, which develop components that can be used in several products and feature teams which develop features specific to a product and releases.

6. Discussion

6.1. Differences to the literature

In this section, the AVL's agile software development framework ALASKA and AVL's agile hardware-/firmware development framework ALADIN will be compared to Scrum from the literature. For comparing ALASKA with the other frameworks, only the team level is considered, due to the similarities. The process, roles, events and artifacts of each framework are examined and compared to the other frameworks to find similarities and difference between them.

The results of this comparison will show the differences/similarities of AVL's frameworks with each other and with the Scrum framework. It will show what kind of roles each framework contains and what responsibilities they share, what events each one has and what their differences/similarities are and it will show the differences/similarities of the artefacts of the frameworks.

6.1.1. Process

Both ALASKA on the team level and ALADIN implement the Scrum framework. Their processes on these levels are similar, but at the beginning, ALASKA and ALADIN start with a vision and a roadmap from which they get their product/team backlog. After that, they will follow the Scrum framework; they start prioritizing the product/team backlog items, select a few of them for the sprint backlog and start implementing them. Every day they will have a daily stand up/Scrum where they meet to discuss what they did and what they want to do. At the end, they will hold a sprint review and a sprint retrospective to gain feedback by presenting a demo. Additionally, they will analyze their processes and try to improve them in the next sprint.

ALASKA is different in this aspect, as its framework is for more than 200 developers. It consists of three different levels. In the first level, the portfolio level, they derive a portfolio strategy containing a vision, a roadmap and a portfolio backlog from the strategies of the ITS department, the segments and the product lines. They also derive a program/product backlog containing also a program/product vision, roadmap and a program backlog. Each program consists of a release plan meeting, which derives the objectives of the program iteration from the program backlog. In order to achieve these objectives, team objectives are created for each team. This helps to establish alignment between the business owners and the program teams. A program itself contains many teams. Each team will start developing after the release plan meeting, and after each sprint a sprint demo and a system demo meeting is held to test the functionality of the whole system. Moreover, Scrum of Scrum meetings are held every week to develop a better understanding of what every team is doing, and every two weeks a program iteration meeting is held to inspect how the progress of the program iteration is going. After these sprints, an inspect and adapt meeting is held which is the same as a sprint review/retrospective. A fully functional program increment can be released after one program iteration.

ALASKA and ALADIN must both be aligned with the product innovation process of AVL.

6.1.2. Roles

Product owner: In Scrum this is just one person who represents the stakeholder and the customer. He or she is responsible for the product backlog and for ensuring that the team delivers value. A difference here is that in ALASKA and in ALADIN the product owner is not only one person. Both frameworks divide the responsibilities into a more market/customer based view (product manager) and into a more development based view (product/development owner). The product/development owner then aligns his or her work with the work of the product manager.

Product manager: is not known (relevant) in Scrum but is essential for AVL. In both frameworks, a new role was established to have a better focus on the market/customer, while also having a good focus on the solution/technologies/team facing view. It is one of the responsibilities of the product manager to be the interface to the market and to the customer, in order to create/maintain the product strategy, vision and the roadmap and to create market-based requirements. In ALASKA, the product manager is also responsible for the program backlog. In both frameworks, the product manager must work together with the product/development owner.

Agile Master: has the same responsibilities as in the Scrum framework, but in ALASKA he or she also has some additional responsibilities. In ALASKA, the agile master is responsible for attending the release-planning meeting to coordinate other agile masters; the system team and other stakeholders. Additionally, he or she must attend a Scrum of Scrum meeting with the other agile master of the program. He or she needs to help estimating larger features and epics in the program, facilitates preparation of the release planning and inspect and adapt meeting and the agile master also has to coordinate the teams under architectural and portfolio governance, system level integration and regular system demos, system level integration and regular system demos.

Development team: In ALASKA and ALADIN they have the same tasks as in Scrum, but in ALASKA the development team is assigned to programs. They collaborate for building increments on the system level instead of smaller shippable increments. In ALASKA, the development teams can also be responsible for features (feature team). They develop specific features for products and releases or for components (component team), where they develop components which can be used for different products.

6.1.3. Events

Backlog grooming: is not a component of the Scrum framework, but is used in ALASKA. Every team member must attend it. In this meeting, they will help preparing the team backlog for the upcoming sprint meeting. This includes adding new stories, adding new epics, breaking down epics into stories and estimating the effort for existing stories. This meeting takes five to ten percent of the time of one sprint but helps the development owner to forecast the content for the next two to four sprints.

Sprint planning: in the Scrum framework, only one sprint-planning meeting exists for one sprint with a time box of four hours for a two-week sprint. This is different to ALASKA and ALADIN, in which the sprint planning is split up into two events, which can be held one after another or at another time. In the sprint plan meeting one, they get together to discuss and forecast what the team could deliver and what work is needed to achieve their goal. All together, they make a pre-selection of the items that shall be implemented. In the sprint plan meeting two, the previous chosen stories are divided into tasks with a maximum length of one day. However, both the AVL frameworks and the Scrum framework do the same things in this meeting.

Daily stand up/Scrum: in both the Scrum framework and in the frameworks of the AVL there is no difference to this meeting. It is a fifteen minutes meeting where all the team members gather together on a daily basis to discuss what they did yesterday, what they are planning to do today and what problems they had/face.

Sprint review: in the Scrum framework and in the AVL frameworks the purpose of this meeting is to reflect on the results, to check what has been “Done” and what has not been “Done” in the current sprint. It takes about four hours for a one-month sprint in which all team members should participate. The difference here is that the product manager also participates even if he or she does not exist in the Scrum framework. A working demo can be demonstrated in this meeting to receive feedback quicker from the stakeholder, to check whether solutions are on track and to make some adaptation on the team/product backlog if necessary. This demo presentation in the Scrum framework is part of the sprint review meeting but in ALASKA the demonstration may constitute an own independent meeting.

Sprint retrospective: The purpose of this meeting is to review what went well and what went wrong in this sprint and to find some aspects to improve for the next sprint. Differences to the literature are that the meeting takes three hours in the Scrum framework, sixty minutes in the

ALASKA framework and ninety minutes in the ALADIN framework. In this meeting, all the team members, which had a role in the sprint, should participate. While the Scrum framework and the ALADIN framework do the same thing in this meeting, ALASKA splits the meeting into two different aspects; the quantitative part in which questions are answered whether they did meet the sprint goal and to review some sprint metrics. The other part is the qualitative part in which they try to analyze the sprint itself on what went well, what did not went well and what they can improve.

6.1.4. Artefacts

Product/Team backlog: Both the Scrum framework and the AVL frameworks use a product/team backlog. There is almost no difference between them, but in ALASKA, the main source of the items in the team backlog comes from the program backlog and from the team's context and from the stakeholder outside the team. It consists of user stories, refactors, maintenance and of architectural stories. To have a good balance between what should be implemented, ALASKA uses capacity allocation to set what should be implement in the current sprint. ALADIN gets its features (according to the Scrum framework) from the market, the stakeholder and other sources. Multiple teams can work on just one product backlog. In ALASKA, the team backlog is available for only for one team.

Sprint backlog/ Scrum board: As in the Scrum framework, ALASKA and ALADIN use a sprint backlog/Scrum board for the selected items from the product/team backlog. These selected items are then modified to deliver a shippable increment.

Shippable increment: Usually, in Scrum a shippable increment would be the sum of all finished sprint backlog items. This increment must be a working piece of the product. While this works well for software, this is not easy to achieve for hardware development. Therefore, ALADIN demands that the increment must not be a piece of the product but rather something you can gain value and feedback from.

Vision/Roadmap: While the Scrum framework does not say anything about having a vision or a roadmap, in reality every company needs some kind of vision and roadmap. It is vitally necessary to have a vision in order to set a goal which the product or the product development must somehow be able to achieve; reflecting stakeholder needs and features which addresses them. The product roadmap is also important for establishing a program to business alignment, while also providing a visibility of what should be delivered at what time. In ALASKA, the roadmap is planned for the six months of program iteration within release milestones.

6.1.5. Comparison agile literature, ALASKA, ALADIN

The table 4 shows the differences and similarities between the frameworks presented in the literature and between the agile development methods of AVL ALASKA and ALADIN. With this it is easier to see the differences and similarities between the practical used agile frameworks from the AVL and the frameworks presented in the literature. Between the literature and the agile development methods of AVL the differences are not so big. Some roles such as the scrum master got more responsibilities, the product manager which is not defined in the literature is used, because of the need to have a role which is responsible for having a constant overview over the market. Artefacts and meetings where are quite similar just adaptations were made in order to meet the needs of the AVL.

	Literature	**ALASKA**	**ALADIN**
Setup	only one team	contains portfolio level with many programs with many teams	one team
Approach	product owner defines requirements	first vision and roadmap then requirements (on all the levels)	first vision and roadmap, then requirements
Focus	software development methods	software development method	hardware, firmware development method
Meetings	Scrum meetings	Scrum meetings + program meetings	Scrum meetings
Product Owner	one person representing customer	one person representing the customer	divided into product owner and product manager
Product Manager	not defined	included can be split into product management and content management	included
Scrum Master	included	included with more responsibilities	same as in Scrum
Sprint/Iteration length	2-4 weeks	2 weeks (team level)/10 weeks (program level)	2-4 weeks
Backlog Grooming	not defined	preparing backlog for sprint planning	not defined but is used
Sprint planning	one meeting four hours/two weeks	split into two meetings six hours/two weeks; same content as in Scrum	split into two meetings four hours/two weeks; same content as in Scrum
Backlogs	product backlog and sprint backlog	program backlog, team backlog and sprint backlog	product backlog and sprint backlog
Shippable increment	functional and shippable part containing finished sprint items	program increment (parts of many sprint increments), sprint increment (functional and shippable part containing all the finished sprint items)	can be anything as long as it brings value and feedback

Table 4.: Comparison of Scrum in the literature with ALASKA and ALADIN

6.2. Interviews

This section will provide an overview of the used method for performing the interviews. For this thesis, expert interviews with the employees of AVL were performed about the topic of agile development in different kind of fields. Therefore, hardware-, software- and firmware developer from the department MIE/MIS were chosen as experts and expert interviews were carried out.

The aim of the interviews was to gather information about their current way of developing products with hardware, software and firmware. With this information, it is possible to see were difficulties arise from using traditional development methods such as the waterfall model or AVL's PIP. Furthermore, the interview's aim was to gain information about the opinion of each employee concerning the usage of agile development methods. They were asked what adaptations need to be done in order to use it for developing products with hardware, software and firmware. With the results from the interviews and with the literature review (chapter2, chapter3 and chapter4), this thesis's research questions (chapter1.2.3) will be answered. Based on the interviews results and on the results of the comparison of AVL's development method with the results from the literature review proposals on how adaptations could be made in order to use agile development for products with hardware, software and firmware.

Methodological approach

Interviews were carried out by using the method expert interviews. Experts are defined by Gläser & Laudel (2010) as people who have special knowledge about social issues. Experts have special privileged access to knowledge about company internal processes, the organization, the own working processes and of their kind of working field. Expert interviews are therefore a special method for collecting knowledge. Gläser & Laudel (2010) define an expert interview as a semi structured interview, which means that there are no standardized answers or questions but rather there are requirements, for performing an interview such as having an interview guide.

In this thesis an interview guide was created (appendix A 3) by applying the theory which was found during literature research. Univ.-Ass. Dipl.-Ing. Harald Wipfler (TU Graz) and Dr. Rüdiger Teichmann (AVL) also contributed to creating the interview guide.

The goal of the interview guide is to structure the interview. This structure should prevent the interviewer from omitting critical aspects, and it should specify the focus of the interview. It should also make a comparison between the other interviews as easy as possible. According to Gläser & Laudel (2010) the questions from the interview guide should be a clear, not to provocative, open and unambiguous. (Gläser & Laudel, 2010)

Table 5 provides an overview of the characteristics of the interviews. Interviews took about 40 to 70 minutes and were performed face to face in German and they were digital recorded in order to evaluate the expert interview in the most precise way. For evaluating the expert interviews, they were structured and the gained information was divided into different categories in order to answer the research questions. This method is described by Gläser & Laudel (2010) as qualitative content analysis. However, instead of transcribing the whole interview only keywords were transcribed and used for answering the research questions.

Employees with different kind of expertise and from different departments of AVL were chosen as interviewees. Six Employees from the department MIE/MIS were chosen because this thesis is created for their departments and because they will use the agile development methods ALASKA and ALADIN in the development of their projects (appendix A 25). These employees are experts in either hardware-, software-, firmware or hardware- and firmware development. Two experts, one for ALASKA and one for ALADIN were chosen to deepen the knowledge gained through the internal documents of ALASKA and ALADIN (appendix A 27). Furthermore, two employees of other departments in the AVL were chosen as best practice since they already use the agile development methods ALASKA/ALADIN in their projects (appendix A 30). With their interviews, it is possible to have a view on the practical results and on their adaption of ALASKA/ALADIN to their needs. They will also show what kind of challenges/opportunity they faced during using agile development methods.

For adjusting and for checking the correctness of the questions from the interview guide one sample interview with a student employee was performed and transcribed (appendix A 9). The student is a software developer from the MIE/MIS department. Based on the gained information, the interview guide and the corresponding questions were adjusted.

Topics	**Characteristics**
Interview method	Expert Interviews with interview guide
Interview evaluation method	Qualitative content analysis
Language of the interviews	German
Interview Length	40-70 minutes
Transcription of the interviews	Keywords
Number of Employees interviewed	Twelve
Experience in their field	One to thirty years
Experience in agile development	No experience in agile development, but all the employees got an enrollment course.
Roles of interviewed employees	• Hardware developer • Software developer • Firmware developer • Project leader • Project coordinator • Department Manager
Distribution of interviewed employees	• Three hardware developer • Three software developer • One firmware developer • One hardware-/firmware developer • One ALASKA expert • One ALADIN expert • Two employees of AVL which already use agile development method

Table 5.: Interview characteristics

Results The results from the interviews are presented in this section. The results are divided into three different parts:

- Interview results from the employees of the MIE/MIS department who will use the agile development methods ALASKA and ALADIN (table 6, table 7 and table 8).
- Interview results of the two expert for ALASKA and ALADIN (table 9).
- Interview results of the employee from the different department which already use the agile development methods ALASKA/ALADIN (table 10).

The results of the interview for the developer of the MIS/MIE department were put together in table 6. The questions for the employees of the MIE/MIS department were more about agile development in general, to get an overview of the expected attitude of the employees. While the questions concerning the others interviewed employees were more about ALASKA and ALADIN than about agile development in general. Most of the employees of the MIE/MIS department are rather positive about using agile development methods in their project, but they were unsure if it would work. They fear that they will fall back into their old behavior and will develop as they did before. The old development method was based on a classical waterfall development method with small adaptations to meet the needs of the AVL. This method was too structured and not flexible; therefore a lot of planning was needed to be ready for changes. For smaller projects this quite an overhead and therefore it was not fully used. This method was structured into different phases. In order to go from one phase to another, everything what was planned for this phase needs to be finished or they would not go to next phase. Problems here arose from stakeholder of the next phases. They did not want to take the responsibility if not everything was to their satisfaction and therefore they did not let them go to the next phase. However, due to this structure, every employee knew what has to be done next.

Topics	**Results**
Opinion	Positive, excited, unsure whether it will work
Weakness of the old development method	Too structured, too much planning, big overhead, no resources, long delays, development hindrances through stakeholder, change of technology, unsure requirements, no decision making, communication was not so good, responsibility was no acknowledged, no focus, unrealistic appointments, not flexible
Strength of old development method	Structured, good planning for bigger projects, clear planning, communication, developing feature by feature
Challenges for adopting agile development	Hardware dependencies, two week sprints for hardware is not realistic, resources, AVL culture, fear of going back to the roots, working shippable increment, testing together
Opportunity through agile development methods	Better structured, more flexible, better focus, better communication, transparent responsibility, clear decision structure, less delays, fix goal, definition of “Done”, knowledge transfer
Overcome challenges through	Usage of prototype/evaluation boards, try and test, more training in agile development, re prioritizing, 4 week sprints

Table 6.: MIS/MIE employee interview results

The employees of the MIE/MIS department stated that if they wanted to use agile development methods, some challenges must be faced in order to fully use agility (table 7). The employees stated that agile development methods are more software development oriented and therefore will not meet all the requirements of hardware development. Hardware development faces dependencies which software development does not face. Hardware development faces dependencies such as procurement for components, development of hardware boards and equipping the components to the boards. The delivery of the fully equipped boards will take several months and therefore it is hard to plan everything. To be agile one has to react to requirement changes but this is not particular easy for hardware development due to its long delivery times. Changes must go through the whole process of development, procurement, equipping and testing again. Additionally, creating a working shippable increment with hardware in a two weeks sprint is not an easy task and mostly impossible. On the contrary, it is rather hard to only develop a small part of the hardware because parts depend on the other hardware parts sometimes. In the end of a sprint, the entire shippable increment must be tested together, but if the hardware is not finished, other ways must be found for testing the software and the firmware together with the hardware.

A challenge which must be also faced is that hardware developers are sometimes firmware developers in the MIE/MIS department, and because they will not only work on one project but rather for many, the development time for each project must be set in a way to be agile and productive. Stress may arise due to the development of a product. The employees of the MIE/MIS department stated that they fear that they will go back to their old development method and their old attitude. They fear that not every developer is happy with having more responsibility and that not every project leader embraces the idea of giving the power to make decision to the developer. Changing the culture will be a great challenge if not every employee accepts the changes in having more or less responsibility. Therefore, a good Scrum master and top-level management support is crucial. Workshops may help the employees to take more responsibility. In addition, it must be clear to every employee who is responsible for what and who makes the critical decision if problems arise through developing the project.

Challenges	**Interviewer**
Hardware dependencies	I1, I2, I4, I5, I7
Sprint length	I1, I4, I5, I7
Resources	I2, I7
Culture change	I2, I3, I6, I7
Shippable Increment	I1, I2, I3, I4, I6, I7
Clear responsibility	I6, I7

Table 7.: Challenges using agile development from interviewer perspective

The employees of the MIE/MIS departments also discussed certain benefits from using agile development methods in their projects (table 8). One major advantages was that it is better structured than the method before, it will not be so strictly structured and no phases will exist but rather they will have a sprint iteration in which a sprint planning is held for planning the requirements. Daily standups ensure that everyone knows how the project is progressing and a sprint reviews/retrospectives to test and to improve their agility. This kind of planning will help them to be more flexible with requirement changes and with unforeseeable problems. The employees stated that having only requirements for a sprint and having a daily standup everyday would help them to have a better focus on developing only things which are needed now. Having fixed definition of "Done" is also a great advantage for the developer of MIE/MIS. Everyone will know if they are finished and what has to be done in order to have a shippable increment in the end.

Benefits	**Interviewer**
Better structured	I1, I2, I3, I4, I5, I6, I7
More flexible	I1, I6
Better focus	I1, I2, I6, I7
Definition of "Done"	I2, I4
Shippable Increment	I1, I2, I3, I4, I6, I7

Table 8.: Benefits using agile development from interviewer perspective

The interviews with the expert were rather a presentation of their framework, than an actual interview. Only questions regarding unsure topics of ALASKA and ALADIN were asked. The information gained from these interviews can be seen in table 9.

The information gained from the interviews with the expert of ALASKA/ALADIN were used for completing chapter 5. Information gained from the interview with the expert of ALASKA was used to completely understand why ALASKA was created, what the difference between content owner and product owner is, what a program is and how they handle requirement changes in their backlogs. Information gained from the expert of ALADIN was used to completely understand why ALADIN is created, how they handle hardware dependencies and why they choose this kind of definition for the sprint increment.

Both interviews also helped to understand what each meeting does and how the planning is done.

Topics	**ALASKA**	**ALADIN**
Idea	ALASKA is not an PIP, needed a framework for managing many teams with different products, Scaled Agile Framework (SAFE)	ALADIN is a little based on ALASKA, idea was to adapt agile development to Hardware development, 4 Pilot projects
Structure	Portfolio level, Program level and Team level based on Scrum	Scrum with some adaptations
Backlog	program backlogs, team backlogs, sprint backlogs	product backlog, sprint backlog
Features	product management and portfolio management defines features, epics are broken down to program features, program features are broken down for each team, than each team breaks down their stories into tasks	product management defines features which will be broken down into tasks
Sprint length	fixed two week sprint	two to four week sprint
Increment	split up into program and sprint increment, sprint increment should be a working piece of the product, program increment is a release of the product	a sprint increment is not a working piece of the product, rather it should create value/feedback

Table 9.: ALASKA and ALADIN interview results

Two employees of AVL from other departments, which have already used agile development methods ALASKA/ALADIN in their projects, were interviewed as best practices for this thesis. The interviews helped to understand how they implemented agile development in their projects, and what challenges they faced and how they were solved. These interviews are used as references for developing an approach for using agile development methods in the MIE/MIS department. The interviews were held with the Scrum master of the current pilot agile team of the MIE/MIS department. The interview was held because of the Scrum master's engagement. The main question was how they implemented agile development and what adaptations were done in order to solve the challenges in adapting agile development methods in hardware, software and firmware development.

Table 10 shows how they adopted the AVL agile frameworks to their needs.

	Best Practice 1	**Best Practice 2**
Method	ALADIN	ALASKA/ALADIN
Requirements	Product management defines them	product management defines and prioritized them with the product owner
Planning	1 year planning with release plan, with milestones and dependencies, two month detailed planning, defining sprint goal, using Scrum of Scrum and backlog refinement	roadmap with goals, sprint plan meeting 1: only with feature owner defining what to do, sprint plan meeting 2: selecting team member and creating tasks; backlog refinement as pre-selection for sprint planning 1; an own product management meeting for defining and prioritizing features
Team	three teams with three to seven developers	one team with twenty-five employees, is split up into feature teams
Backlog	one backlog for three to four week sprints	one backlog with smaller backlogs for each four teams, four Scrum boards with detailed(current and next sprint) and rough planning(two quarter)
Sprint length	two weeks	three weeks

Table 10.: Best practices from the AVL evaluation

6.3. Challenges concerning the practical use of agile development methods

ALASKA (software development) and ALADIN (hardware-/firmware development) are special frameworks for either hardware/firmware or software, but for a product which needs hardware, software and firmware there must be some way to align these frameworks. Usually problems arise when different frameworks have to work together. In this section, I will discuss some of the main problems that can be seen by comparing the frameworks among themselves and by comparing them with the literature. Furthermore, interviews were held with the employees of the AVL to include their opinion on what problems may arise and how they can be solved. The challenges from the comparison and from the interviews are discussed in this section.

The challenges identified through the interviews and the literature review were:

- Hardware dependencies
- Working shippable increment
- Demo
- Testing
- Resources
- Culture change
- Responsible employee
- technology change

The other challenges identified through comparing ALASKA with ALADIN and with the literature were:

- Process
- Vision/roadmap
- Product/team backlog
- Communication
- Roles

Process: ALASKA and ALADIN are two different processes. Even if they are quite similar on the team level, the other levels differ from ALADIN. ALASKA is also a framework for more than 200 developers. Having a product which needs both frameworks is problematic due to the dependencies of the team backlog on the program backlog in ALASKA. Epics which are split up into two different frameworks are hard to handle. They must be discussed in the program release meeting of ALASKA and they need to be prioritized. Depending on how many program epics exist and what features are more important for this program iteration, it could lead to a major loss of time.

Alignment of the sprints is also a problem between ALASKA and ALADIN. In ALASKA, the sprint takes about two to four weeks, while in ALADIN it only takes two weeks and there are also other iterations in the program level which take ten weeks.

Hardware dependencies: The main problem with having a product which needs hardware, software and firmware is that there are hardware dependencies. Hardware is bound to the procurement, suppliers and current hardware technology and it takes some time to develop products. Changing parts or fixing some problems takes time and money and this might lead to loops in planning, designing, buying the parts and creating the layout. Without any hardware, it is also a problem for the firmware itself. How should a firmware developer develop something without having a prototype?

Vision/Roadmap: ALASKA and ALADIN both have a vision and a roadmap, but in ALASKA, these vision or roadmap derives from the product line and from the product strategy and there are other visions and roadmaps for the whole portfolio which contains all the programs itself. ALADINs vision/roadmap is solely derived from the product manager and his product strategy. An alignment of these visions and roadmaps must be done in order to avoid problems concerning prioritizing the values.

Working shippable increment: The Scrum frameworks demands that after each sprint a working shippable increment must be created. This increment must be a fully functional piece of the product. This works well for software but will not work for hardware. Creating just a piece of the product is rather hard for hardware and it would also take much longer, due to planning and designing the layout, ordering the parts, creating the prototype and testing it before. The hardware development employees all raised the same question, namely how to develop hardware in a just two weeks of iteration with a working increment at the end. Most of the employees fear that this might not work, since it takes about two to three month to have a prototype or a piece of hardware ready.

Demo: An important aspect of Scrum is to have a demo after every sprint which can be presented to the stakeholder to gain feedback, but this is rather difficult for a product which has hardware dependencies. Hardware cannot always be developed in just two weeks. Therefore, there must be another way to receive feedback from the stakeholder.

Testing: Testing software, hardware or firmware by itself is not difficult to do, but how should firmware be tested if there is no hardware. It takes some time to get a prototype delivered in order to test hardware and firmware together. This might last a few months, which could cause a delay in delivering the product. Other ways must be found to test hardware and firmware without solely depending on a prototype.

Product/Team backlog: A problem might be that when both, ALASKA and ALADIN, would have an own product/team backlog, progress cannot be measured as with just one backlog, and the transparency might suffer. Furthermore, should there be only one product backlog for each product or should there be one product backlog for all product?

Communication: For having a good communication and information exchange the teams in ALASKA and ALADIN need to either attend the same meetings, which would need alignment of the meetings or have an own Scrum of Scrum. Without these, misunderstandings and delays may occur.

Resources: Another crucial aspect for developing a product is to have enough workers. ALASKA and ALADIN could not help developing a product faster if the needed workers would not be there. The employees feel that they will not have any time for other products anymore. Therefore, there must be a way to plan how much time each employee has for which product. A good balance must be found.

Roles: Having two frameworks means also having a Scrum master for each framework and perhaps two product owners or development owner. They also have to communicate together a lot in order to establish a profound understanding of what the other team needs how they are proceeding.

Culture change: All the employees were positive towards implementing an agile development method, but most of them feared that changing the culture could be difficult. They claimed that it would take some time and effort to change the mindset of the employees and of the management and that the developer would be responsible for planning what to do next and how they should do it.

Responsible employees: Having a product manager, project leader and some other managers, the responsible person for making a decision is not always visible, therefore some employees said that at some point when a clear decision was needed, it was not clear who the responsible person was and discussions between the management, the project leader and the product management occurred and this caused to an enormous delay.

Technology change: Changing technology while being in the middle of a project could lead to delays resulting from adapting the new technology and learning how to work with it.

6.4. Proposals

In this section the results from the literature review, from comparing ALASKA and ALADIN and from the interviews (mostly from the opinions of the employees and from the best practices from AVL), are put together to create few proposals on how to solve the challenges from the previous section 6.3. It will also be described how specific roles such as the Scrum master, the development owner and the teams should be handled and how meetings should be held with which member.

The proposals are split up into different parts:

- **General:** Proposals are presented for challenges which will, regardless of whether a combination of ALASKA and ALADIN or ALASKA and ALADIN as separated frameworks or downgraded version is used, will occur.
- **Combining ALASKA and ALADIN:** Proposals are presented for the usage of this method and for the challenges which occur when combining ALASKA and ALADIN to one framework.
- **ALASKA and ALADIN as separated framework:** Proposals are presented for the usage of this method and for challenges which occur when using ALASKA and ALADIN as separate frameworks.
- **Using a downgraded Version:** Proposals are presented for the usage of this method and for challenges which occur when using a downgraded version of ALASKA with some adaptations from ALADIN.

6.4.1. General

Hardware dependencies: Employees stated in the previous section 6.3 that there are many hardware dependencies which need to be solved in order to implement agile development such as: shipping time, part procurement, testing, etc. Developing a product based on hardware, software and firmware always leads to dependencies. A way of reducing dependencies of the hardware would be to have some evaluation boards, prototype boards and simulation for testing the hardware and firmware until a prototype arrives. Having an agile development method for hardware could also help to reduce the error rate in the prototypes by gaining feedback faster from customer/stakeholder and by showing some concepts and designs. Additionally, delivery time could be reduced by having employees from the sales/procurement attending meetings such as the release plan meeting. Important key parts could be ordered earlier. This way only some parts are left.

Culture change: Changing the culture is a rather difficult task. Employees feared that if they use agile development, the management would not accept the shift of responsibilities. Therefore, the mindset from the management must be changed. Due to Scrum, the responsibility and the decision-making is shifting and then the mindset of the employees must be changed. The employees must be aware that they are responsible for making decisions on how to develop. This will take some time and it will need the help from the top management and from a good Scrum master.

Scrum/Agile team: The Scrum team must not only consist of the developer, the Scrum master and the product owner/product manager but of mechanics, procurement employees, suppliers and testers. This reduces unnecessary meetings and communication and leads to a reduction of development/delivering time.

Responsible employees: Employees were not always sure which person should be asked to make a decision and to take responsibilities at some point. Therefore a clear decision structure is needed in which it is easy to see who is responsible for making a decision for different questions regarding requirements, changing requests and technology changes.

6.4.2. Combining ALASKA and ALADIN

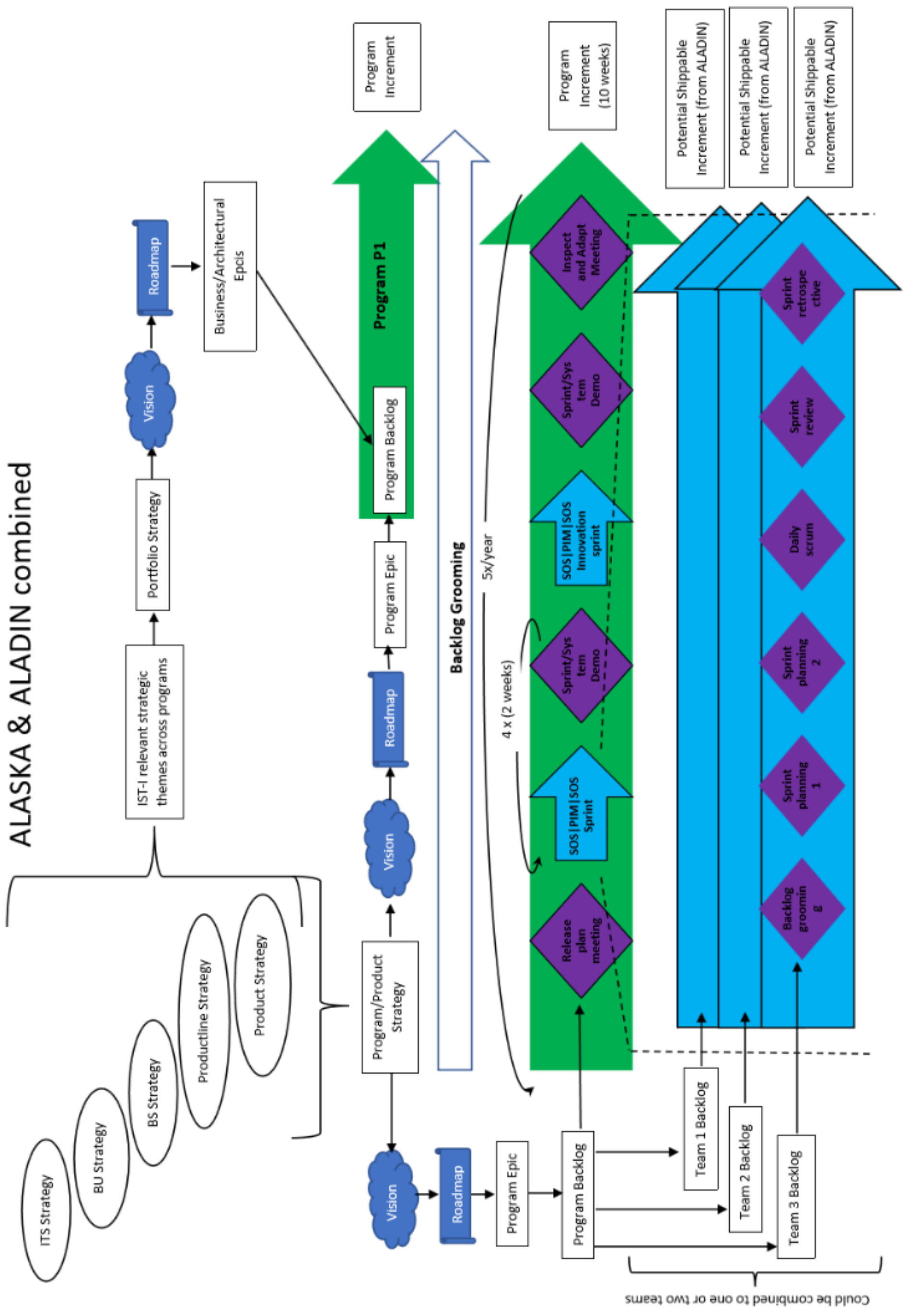


Figure 16.: ALASKA and ALADIN combined to one framework (personal design based on: AVL, 2017, 2018)

Because ALASKA and ALADIN are not especially different from each other, they could be combined to one framework (figure16). In most cases it would be recommendable to adapt ALASKA as the main framework because ALASKA is already implemented in many other departments. To do this, some things need to be changed or taken from ALADIN.

Development team: There are two possibilities for the development team: firstly, there should be only one team consisting of software, firmware and hardware developer. All the knowledge needed for developing the products should be in the team. Other possibilities would be to use more than just one team. The teams could be separated into a hardware, firmware and software team or depending on the needs.

Scrum master: For the combination of both frameworks, only one Scrum master is needed. Even if there was more than just one team.

Product owner/product manager: For each product, a product owner/product manager is needed. Even if there is only one team or more teams, the teams could be responsible for different products and therefore it is recommended to have different product owners/product managers, which discuss together what to prioritize in the next sprint and which task a higher priority has.

Development Owner: Depending on whether there is only one team or more, more development owners are needed to take the responsibilities for managing the backlog.

Backlog: For the backlog, there could be one program backlog for all the teams. These program backlogs should be filled and prioritized from the product manager. The backlog from each team is then derived from the prioritized items of the program backlog.

Shippable increment: Due to the literature review and the interviews it is clear that it is not always possible for hardware development to have a functional shippable increment. This definition must be adapted from ALADIN, where a shippable increment could be anything which would gain some feedback from customers and other stakeholders. At the beginning the shippable increment could be split up in a software, firmware and hardware part until some hardware is available.

Release planning: In the release-planning meeting, the time for ordering and receiving the prototype needs to be considered, as a demo without hardware would not be useful. Therefore, he time developing hardware, software, firmware, ordering the parts, creating layout, equipping the layout, simulating hardware and delivering time need to be considered.

Meetings: Meetings would consist of all the hardware, software and firmware employees or just the teams for the daily standups. This must be tested in order to see which possibility would be better.

Testing: This is one of the most difficult parts, as for testing the whole product hardware, firmware and software must be already finished. However, it takes some time to obtain a hardware prototype, since firmware and software cannot be tested before the prototype is ready.

For testing firmware on a lower level, buying some evaluation boards or prototype boards, which can be obtained faster, might be a solution. In the end, it would only test the functionality but not the dependencies with the real hardware. Therefore, having a prototype is essentially necessary at a later point. For firmware without any hardware dependencies, some test functions could be developed to test them.

Testing hardware without having a prototype is also not particularly trivial. A solution might be a form of simulation. Hardware could be simulated and tested, also some parts could be tested by buying hardware parts can be bought which are similar to the ones ordered but which can be obtained faster. However, having a finished prototype at a later point of the development is also essential.

For testing software, there are not that many hindrances, as software can be tested independently from hardware or firmware at the beginning with the help of some test functions.

After the prototype is available, the whole functionality of the product and the hardware dependencies can be tested together.

6.4.3. ALASKA and ALADIN as separated frameworks

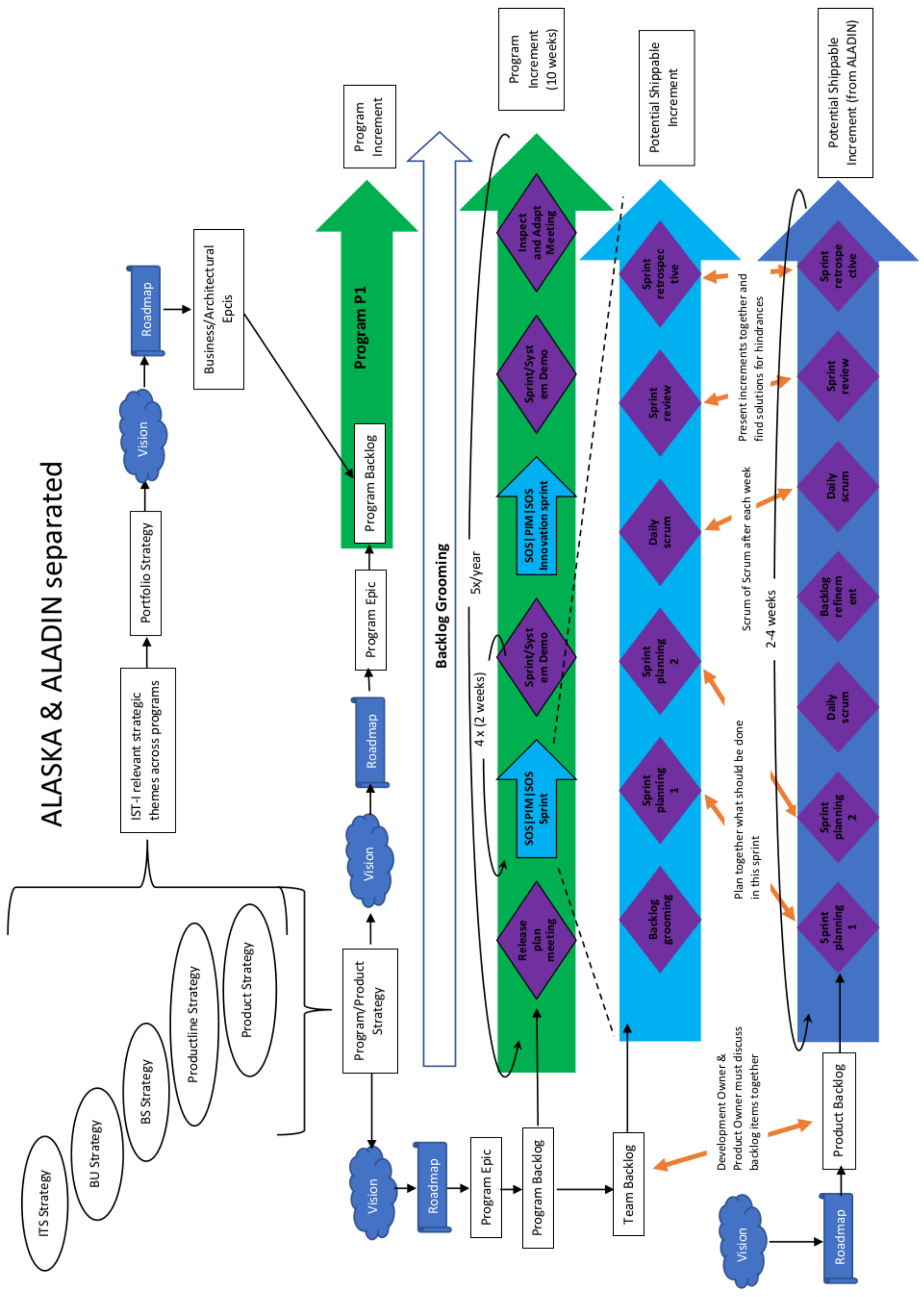


Figure 17.: ALASKA and ALADIN as separated frameworks (personal design based on: AVL, 2017, 2018)

Choosing to have two different frameworks for one product could lead to some problems. Alignment must be done for the sprints, sprint demos, vision, roadmap, backlog, etc. In figure 17 both ALASKA and ALADIN are displayed, the interaction between these two frameworks are here marked with orange arrows. As the interaction between the team backlog from ALASKA and the product backlog from ALADIN, the difference between these backlogs is that ALADIN does not define an own backlog for each teams, rather they have just one backlog for multiple scrum teams. In ALASKA there is an own backlog for each team, which gets its items from different sources such as from the program backlog.

Scrum master: A decision must be made whether there is one Scrum master for each framework or not. Having one Scrum master might be problematic, as he must attend the meetings of ALASKA and of ALADIN. This might be a lot of work. Additionally, the Scrum master must split his or her time to have equal time for ALASKA and for ALADIN. Having two Scrum master would lead to alignment problems. Hindrances which need to be solved could involve both of them. Moreover, they have to communicate together a lot in a Scrum of Scrum.

Product owner/Product manager: Here the same applies for the product owner/manager as with combining ALASKA and ALADIN. There is more than just one product owner/manager and they must discuss what should be prioritized higher and what should be taken into the backlog.

Backlog: For the backlog there would be one program backlog and team backlogs for ALASKA and one team backlog for ALADIN. The team backlog of ALADIN could be derived from the program backlog of ALASKA.

Vision/Roadmap: The vision and the roadmap for the product must be the same for each framework. Therefore the product manager of ALASKA and ALADIN must communicate what they want to do with this product. Having two different vision/roadmap might lead to different priorities for the development of the product.

Sprint: Depending on how long the sprint takes in ALADIN, an alignment must be done in a way that after each sprint, increments can be developed and some testing can be done together. A good way to do this would be to choose two or four weeks for ALADIN. This way they would be aligned, every fourth week they could develop an increment and test together and every second week the software team could test their increment on their own.

Strategy Alignment: In ALASKA there are two different strategies: the portfolio strategy is the same for every program and the product/program backlog. In ALADIN there is only a product strategy. In order to agree on what is important, all the strategies must be somehow aligned or priority issues might occur.

Meetings: Other decisions must be made concerning the interviews, such as sprint planning 1 and 2, daily Scrum and backlog refinement. Having two different frameworks also means having twice as much meetings. The question arises whether the meetings should be combined together or whether they should be held separately for each framework. Having a combined meeting would help both teams to have a better understanding of what the other team is doing and what is still needs to be done. By having separated meetings, the communication and the knowledge transfer between the teams may suffer. Other meetings (Scrum of Scrum) must be implemented to share information between the frameworks. In most cases, these would be held either weekly or biweekly.

Shippable increment/program increment: Having separated frameworks would also mean to have two separated shippable increments; the hardware increment would be not a finished prototype at the beginning but rather a concepts or designs. This would be the same as in the combined proposal. In the program increment where all shippable increments of the programs are combined, the hardware could also be integrated if the hardware is available.

6.4.4. Using a downgraded version

Another possibility would be to use a downgraded version of ALASKA 18. ALASKA is a good framework for about more than 200 developers, but it creates a lot of overhead for smaller numbers. Therefore, it is recommended to only use the team level with some practices from the program level (figure18).

Scrum Master: Only one Scrum master would be needed for the whole framework.

Product owner/product manager: It is essential to have a product owner/product manager. They also need to discuss (as in the other two proposals) the backlog and they need to prioritize the items.

Development team: There are again two possibilities. Firstly, there may only be one team containing hardware, software and firmware developers, or they can again be split up in hardware, software and firmware needs. However, teams can also be divided in teams depending on the current needs.

Backlog: One backlog might contain all the prioritized items from the product manager. From this backlog, the backlogs for the different teams can be derived.

Development Owner: Depending on how many teams there are, there must be a product owner for every team who is responsible for managing the team backlog and the team itself.

Sprint: Depending on how the team is split, the sprint length can vary from two weeks for software developer to four weeks for hardware developer. However, in the end every two weeks an alignment can be done in which the increments are presented.

Meetings: Meetings can be held with all the teams together for all the meetings or all teams for the bigger meetings, such as sprint planning 1 and 2. Additionally, it is possible to split up into teams for smaller meetings such as the daily standup. This must be tested to see which one would work better.

Shippable increment: The definition of a shippable increment could be adapted from ALADIN. It is not always possible to present an increment that works, but concepts and other things could be indeed presented in order to gain feedback.

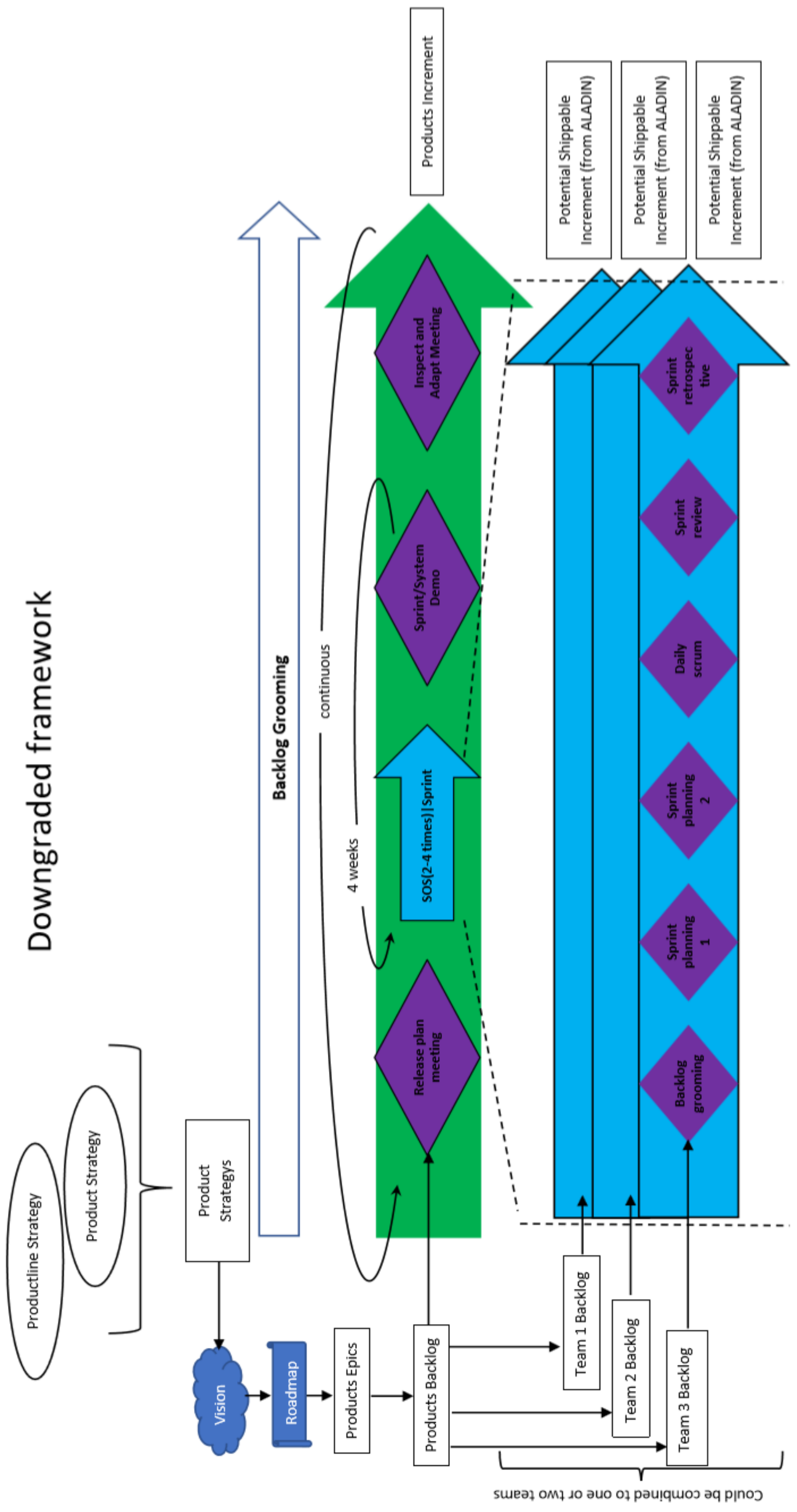
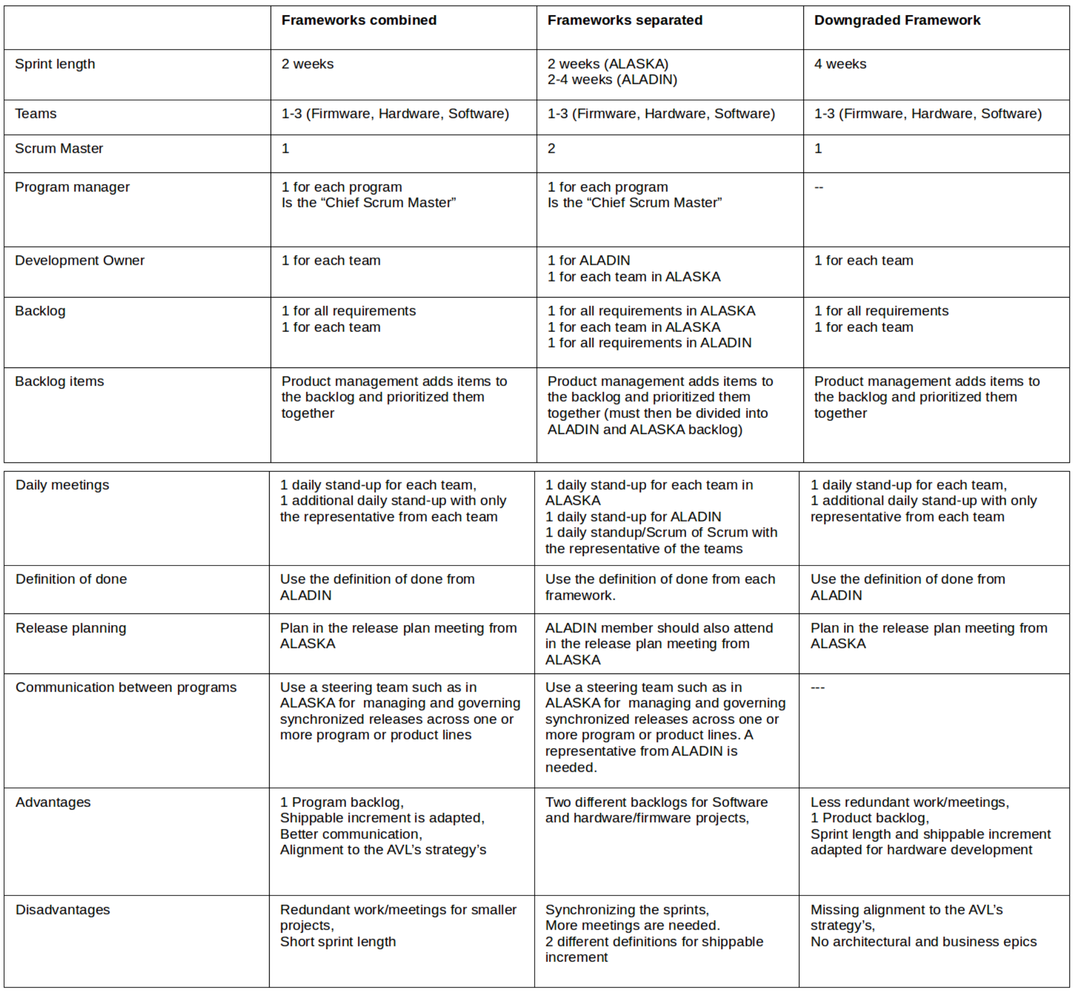


Figure 18.: Downgraded version of ALASKA and ALADIN (personal design based on: AVL, 2017, 2018)

6.4.5. Comparison of proposals for using AVL's frameworks

The figure 19 shows the comparison between the proposals. With this it is easier to see the differences and similarities from the proposals. Agile development areas such as meetings, artefacts and roles, can be compared and the best suitable proposal can be chosen. Furthermore, this figure also enables one to see the advantages and disadvantage in using the proposal.



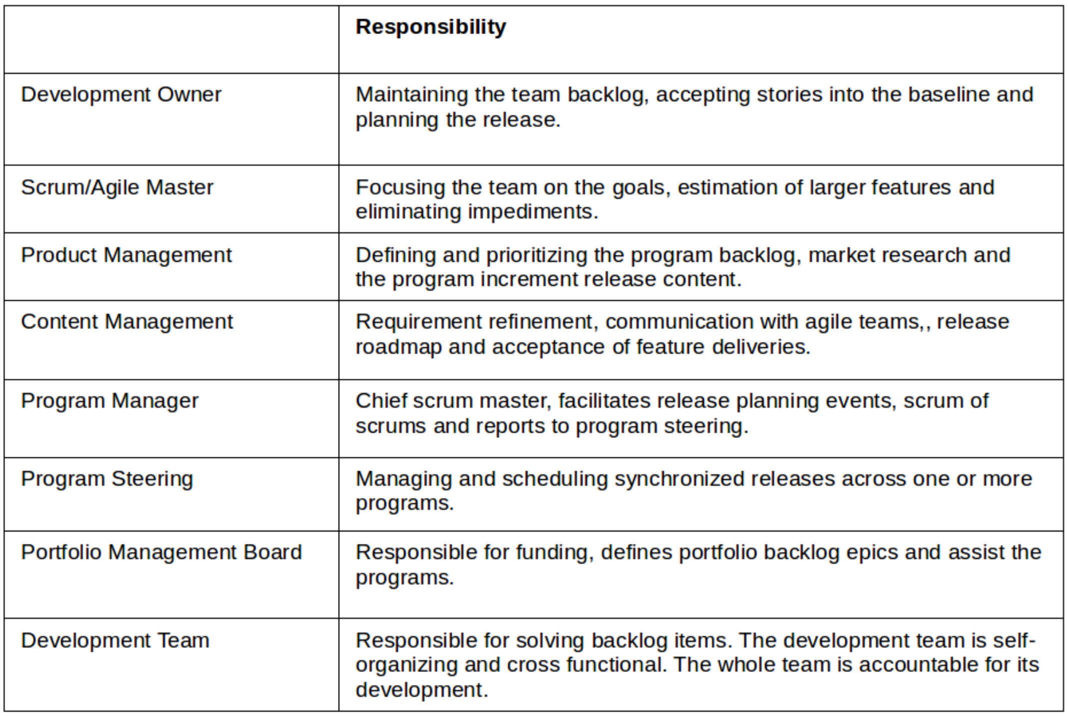
	Frameworks combined	**Frameworks separated**	**Downgraded Framework**
Sprint length	2 weeks	2 weeks (ALASKA) 2-4 weeks (ALADIN)	4 weeks
Teams	1-3 (Firmware, Hardware, Software)	1-3 (Firmware, Hardware, Software)	1-3 (Firmware, Hardware, Software)
Scrum Master	1	2	1
Program manager	1 for each program Is the "Chief Scrum Master"	1 for each program Is the "Chief Scrum Master"	--
Development Owner	1 for each team	1 for ALADIN 1 for each team in ALASKA	1 for each team
Backlog	1 for all requirements 1 for each team	1 for all requirements in ALASKA 1 for each team in ALASKA 1 for all requirements in ALADIN	1 for all requirements 1 for each team
Backlog items	Product management adds items to the backlog and prioritized them together	Product management adds items to the backlog and prioritized them together (must then be divided into ALADIN and ALASKA backlog)	Product management adds items to the backlog and prioritized them together
Daily meetings	1 daily stand-up for each team, 1 additional daily stand-up with only the representative from each team	1 daily stand-up for each team in ALASKA 1 daily stand-up for ALADIN 1 daily standup/Scrum of Scrum with the representative of the teams	1 daily stand-up for each team, 1 additional daily stand-up with only representative from each team
Definition of done	Use the definition of done from ALADIN	Use the definition of done from each framework.	Use the definition of done from ALADIN
Release planning	Plan in the release plan meeting from ALASKA	ALADIN member should also attend in the release plan meeting from ALASKA	Plan in the release plan meeting from ALASKA
Communication between programs	Use a steering team such as in ALASKA for managing and governing synchronized releases across one or more program or product lines	Use a steering team such as in ALASKA for managing and governing synchronized releases across one or more program or product lines. A representative from ALADIN is needed.	---
Advantages	1 Program backlog, Shippable increment is adapted, Better communication, Alignment to the AVL's strategy's	Two different backlogs for Software and hardware/firmware projects,	Less redundant work/meetings, 1 Product backlog, Sprint length and shippable increment adapted for hardware development
Disadvantages	Redundant work/meetings for smaller projects, Short sprint length	Synchronizing the sprints, More meetings are needed. 2 different definitions for shippable increment	Missing alignment to the AVL's strategy's, No architectural and business epics

Figure 19.: Comparison of proposed processes (personal design)

6.4.6. Responsibilities of the roles in the proposals

The figure 20 shows the responsibilities of all the roles within the proposals. No differences were made between the proposals for the roles. Because in all the proposal the roles were quite similar, except for roles from the program and portfolio level. The responsibilities presented are taken from ALASKA and ALADIN. For better understanding, some definitions such feature and component team were put together as development team.

Each roles has a small portion of responsibilities but the accountability must realized in such a way that every role feels responsible for the success of the project. There is no specific role which is responsible for success of the whole project.



	Responsibility
Development Owner	Maintaining the team backlog, accepting stories into the baseline and planning the release.
Scrum/Agile Master	Focusing the team on the goals, estimation of larger features and eliminating impediments.
Product Management	Defining and prioritizing the program backlog, market research and the program increment release content.
Content Management	Requirement refinement, communication with agile teams,, release roadmap and acceptance of feature deliveries.
Program Manager	Chief scrum master, facilitates release planning events, scrum of scrums and reports to program steering.
Program Steering	Managing and scheduling synchronized releases across one or more programs.
Portfolio Management Board	Responsible for funding, defines portfolio backlog epics and assist the programs.
Development Team	Responsible for solving backlog items. The development team is self-organizing and cross functional. The whole team is accountable for its development.

Figure 20.: Responsibilities of the roles in the proposals (personal design)

6.5. Answers to Research Questions

The research questions defined at the beginning of this thesis will be answered in this section based on the literature review, the investigation of the AVL's agile development frameworks and based on the interviews.

- Is it possible to use agile software development methods for products with hardware and

firmware development?
- Does the agile software development method need some adaptations in order to be used in hardware and firmware development, and what needs to be changed?

6.5.1. Is it possible to use agile software development methods for products with hardware and firmware development?

Based on the literature's investigation in section 4, (presenting practical examples of companies which already use agile hardware development) the answer would be yes. Companies from different sectors are already using agile hardware development methods successfully. However, to use agile software development methods for products with hardware and firmware development, some adaptations to the framework must be done. Additionally, it depends on which framework is used. In the literature, Scrum was mostly used as a base framework and adaptations were made to it to meet their needs such as changing the definition for the increment or using bigger sprint length.

Currently the MIE/MIS department from the AVL uses the agile development method ALASKA for software development and for firmware development. Many employees stated that after using this framework many uncertainties, communication and resource issues were solved (table 6). Including the hardware development to this framework or using the agile hardware development methods ALADIN would be the next step. Currently this is not the case and therefore it is not possible to draw conclusions from that. Lastly, it must be said that products using hardware must be planned before in such way that it meets delivery appointments and that it reduces uncertainty regarding the procurement and the production time.

6.5.2. Does the agile software development method need some adaptations to be used in hardware and firmware development, and what needs to be changed?

Because of the nature of hardware development, it is necessary to adapt the framework to the needs. This can be seen in the literature review for hardware section 4.2 and for the firmware in section 3.2 in which a study was conducted in order to investigate what adaptations needs to be done and where different company has stated why and how they changed the chosen framework to their needs to achieve agility.

Hardware development is bound to some physical constraints. These constraints must be considered and therefore the framework must be adapted. However, the basic structure of the framework, such as meetings and roles, may be the same. Firmware does not really need any adaptation from the framework section, but adaptations must be made because of the dependencies from the hardware.

Based on the literature review sections 4, section 3 and the investigation of the hardware/firmware development framework ALASKA section 5 of the AVL it can be concluded that some adaptations (section 6.4) must be done in order to achieve agility.

The main take aways for the AVL would be:

- Depending on their needs the team can consist of hardware, software and firmware developers or there can be separated teams for each one of them.
- Sprint lengths may vary from each team; therefore alignment must be done to ensure that teams can review the increments together.
- The Backlog may consist not only of one team and one product item, but also rather of many teams and many products. These items would be applied into the team backlog afterwards.
- There must be a product manager for each product who knows what the market needs. Because there is not only one product manager but rather several product managers, they need to discuss in a meeting what requirements are needed and what priority they have. These requirements are then put into the big backlog for all teams.
- Increment must not be some working piece of the product itself, but it can consist of concepts, ideas or designs. These can be presented to gain feedback from the stakeholder.
- A release plan must be created to ensure that the product is finished with the requirements needed until the deadline.

7. Conclusion and Outlook

The purpose of the thesis was to investigate whether it is possible to use agile software development for products with hardware, firmware and software development for the department MIE/MIS of the AVL. To investigate what adaptations have to be made in order to use agile software development methods and how the current agile development frameworks of AVL can be used for the development of products with hardware, firmware and software.

A literature review was performed for the fields of agile hardware development, agile firmware development and agile software development. In the field of agile software development methods, popular agile development methods such as Scrum, XP and FDD were investigated, analyzed and compared with each other (section 2). For the fields of agile hardware (section 4) and firmware development (section 3), a literature review was performed to investigate what agile software development methods could be used for hardware and firmware development and how the frameworks needs to be adapted. Practical examples of companies which already use agile development methods for hardware and firmware development were investigated. For both hardware and firmware development, three best practices, were investigated. With these best practices it is possible to find a solution to the challenges that occur by adopting agile software development methods.

Each of the best practices states that at the beginning there were problems with adopting agile software development methods for to their needs. The most popular agile software development methods in these cases were Scrum. Adaptations had been done to the framework in order to meet their needs. These adaptations entail having a bigger sprint length, changing the shippable increment definitions, using models and concepts to gain feedback from customer and including procurement at the beginning of the project. The best practices did not explicitly show how their framework looked after the adaptations, but they rather stated that some events or definitions were not used because of the overhead and the frustration of the employees. However, in the end the implementation and the adaptation of the framework were a success; they were able to develop hardware and firmware with agile development methods.

AVL's agile development frameworks, ALASKA for software development and ALADIN for firmware and hardware development, were also analyzed (section 5 and compared with each other and with the Scrum framework from the literature. ALASKA is a software development process for about 200 software developers. It is a scaled version of Scrum and the AVL currently uses it for their software projects. ALADIN is a hardware and firmware developed framework, which is currently in development and tested through some pilot projects. ALADIN looks similar to Scrum at the first sight, but adaptations were made in order to meet their needs, such as changing the definition of working shippable increment to something that brings value and feedback from the customer (e.q. concepts and designs).

The comparison between ALASKA and ALADIN with the literature (section 5.3 and section 6.1) analyzed the different frameworks and their meetings, artifacts, roles and differences or similarities. The results of the comparison the differences and similarities between the frameworks. It shows that there were only minor differences between each framework (section 6.1. Most of the differences occurred due to the different development fields and their practical use of the frameworks, such as having a product manager which is not stated in the Scrum framework itself, having a release plan with different milestones and market releases, changing the responsibilities of the roles such that the framework works with more than one team and with larger teams.

Expert interviews with the employees of the department MIE/MIS, with employees from other department which already use agile development methods in their project and with experts from ALASKA and ALADIN from AVL were performed. The expert interviews with the employees of the MIE/MIS department show their opinion concerning using agile development methods for projects. Most of the employees are looking forward to using an agile development method (section 6.2) in their projects are and what kind of benefits/challenges they see in comparison with the old development style. Advantages are having more structure, better communication, a fixed definition of “Done”, a transparent perspective on responsibilities and more frequent feedback from customer and stakeholder. The greatest fear was that the agile frameworks would only be a temporary solution and that after some weeks it would not be practiced anymore.

The results from the interviews of the employees from other departments from AVL, which already use agile development methods, show what kind of adaptations had to be done in order to fit the agile development methods ALASKA/ALADIN to their needs. These interviews were performed with the Scrum master of the department MIE/MIS in order to gain information about how other departments handle the situation of developing hardware, firmware and software agilely. In these interviews, their adapted frameworks were discussed and solutions to challenges were presented. It was not like the interviews with the employees from the MIE/MIS department, it rather was a presentation of their methods for developing their products. The gained knowledge is used for creating the proposals and for discovering challenges in this thesis.

The interviews with the expert from ALASKA and ALADIN were used for answering questions regarding ALASKA and ALADIN. The results were used for completing section 5.1 and section 5.2. Most of the questions were asked to understand why the framework was created, how they handle requirements changes in the middle of the iteration and what kind of responsibilities do the roles in their frameworks have and why.

The aim of this thesis was to find challenges that occur by using agile development methods for projects with hardware, firmware and software development and to make proposals for how the agile development methods of AVL can be combined; can interact with each other. Therefore, three proposals for using their frameworks were made. These was done by using the results of comparing AVL's agile development methods, the results from the literature and the results from the expert interviews.

- Combining ALASKA and ALADIN to one framework (section 6.4.2)
- Using ALASKA and ALADIN as separated framework (section 6.4.3)
- Using a downgraded version of ALASKA with adaptations to meet the needs of hardware, firmware development (section 6.4.4)

Using a combination of ALASKA and ALADIN would reduce communication interfaces between these two frameworks. Overhead concerning having too much meetings, synchronizing the different sprint length and having two Scrum master would fall out at team level, while having the program and portfolio level meetings stay. Adaptations in form of changing the definition of working shippable increments to having something which brings value to the customer and helps gaining feedback quicker.

By using ALASKA and ALADIN as separated framework, communication interfaces must be defined. A close communication from the Scrum masters and from the product owner and development owner is crucial for successfully using separated frameworks. There have to be meetings to check the progress of the development, such as having a Scrum of Scrum after each week. Sprint lengths of the frameworks must be fixed. For ALASKA a two week sprint is recommended and for ALADIN either two weeks or four weeks sprint length. This ensures that the frameworks can be synchronized. After four weeks, a sprint review and a retrospective should be done with each member of the framework, in order to test the entire product.

Using a downgraded version of ALASKA with adaptations, would combine the benefits of having one framework, such as the combination of ALASKA and ALADIN. This would also reduce the overhead with having program and portfolio meetings, roles and artefact. Adaptations must be done in order to fit the frameworks to the needs of hardware development, such as having a fixed sprint length of four weeks and changing the definition of a working sprint incremental into having an increment which provides value to the customer and frequent feedback quickly.

The main conclusion of this thesis and the answers to the research questions are that it is possible for products with hardware, firmware and software development to use agile development methods. The thesis shows that the literature already mentions companies who use agile development methods with developing hardware, firmware and software. Most of these companies state that they benefit from it. There are the following benefits:

- Increased cooperation and communication
- More structured work

- Better definition of requirements and better prioritization
- More frequent feedback from different stakeholders
- Fixed definition of “Done”
- Transparent process flow and responsibilities

However, it needs to be considered that due to the nature of hardware development, there are certain constraints which cannot be ignored. The literature mentions that adaptations must be performed in order to use agile software development frameworks such as:

- Depending on their needs the team may consist of hardware, software and firmware developers or there can be separated teams for each one of them.
- Sprint lengths may vary for each team, therefore alignment must be done to ensure that teams can review the increments together.
- The Backlog may consist of not only one team and one product item, but it rather consists of many teams and many products. These items can be derived into the team backlog.
- There must be a product manager for each product who knows what the market needs. Since there is not only one product manager but many, they must discuss in a meeting what requirements are needed and what priority they have. These requirements are then put into the big backlog for all teams.
- An increment must not be some working piece of the product itself, but it should rather be a concept, an idea or a design which can be presented to gain feedback from the stakeholder.
- A release plan must be created to ensure that the product is finished with the requirements needed till the deadline.

Further investigations: It would be interesting to investigate and research further into this topic. However, due to the lack of time and in order to reduce the scope of this thesis, these investigations need to be conveyed in another thesis.

Firstly, it would be interesting to consult more literature for existing frameworks and processes for the development of hardware, firmware and software products. It would also be interesting to have expert interviews with other companies in Austria and with other departments of the AVL which already use agile development methods for products with hardware, software and firmware development. This would show how they handle hardware development constraints and how they solved these challenges.

It would also be interesting to see how the proposals for the AVL were be implemented and what adaptations had to be done in order to solve certain challenges. After some time it would be interesting to have expert interviews again, to see how the opinions of the employees from the department MIE/MIS changed ; how the development changed concerning the initial development methods and the proposed agile development method, and what improvements could be done in order to make the framework more efficient. In addition, it would be interesting to know how big the knowledge transfer in a collocated team is and it would also be of interest to see if it is big enough to let a software developer handle minor hardware development requirements and vice versa.

References

Alliance A., 2013: *The Agile Manifesto*. Accessed: 15.08.2017. https://www.agilealliance.org/agile101/the-agile-manifesto

AVL, 2017: *Internal document for ALASKA*. Accessed: 18.12.2017. https://desktop.avl.com/corp/02/0058/ALASKA/Pages/Default.aspx

AVL, 2018: *Internal document for ALADIN*. Accessed: 18.12.2017. https://desktop.avl.com/corp/02/0087/Pages/Home.aspx

Backblaze Inc., 2015: *Application of Scrum Methods to Hardware Development*. Accessed: 20.05.2017. https://www.backblaze.com/blog/wp-content/uploads/2015/08/Scrum-for-Hardware-Development-V3.pdf

Beck K., Andres C., 2004: *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Boston, 2 Edition.

Eklund U., Bosch J., 2012: *Applying Agile Development in Mass-Produced Embedded Systems.*, in: *XP*, Volume 111 of *Lecture Notes in Business Information Processing*, Springer, p. 31–46.

Gläser J., Laudel G., 2010: *Experteninterviews und qualitative Inhaltsanalyse als Instrumente rekonstruierender Untersuchungen*, VS, Verl. für Sozialwiss., 4 Edition.

Greene B., 2004: *Agile Methods Applied to Embedded Firmware Development*, in: *Agile Development Conference*, IEEE, p. 71–77.

Highsmith J., 2002: *Agile Software Development Ecosystems*, Addison-Wesley Longman Publishing Co., Inc., Boston, 1 Edition.

Holtsnider B., Wheeler T., Stragand G., Gee J., 2010: *Agile Development and Business Goals*, Elsevier, 1 Edition.

Huang P.M., Knuth A.A., Krueger R.O., Garrison-Darrin M.A., 2012: *Agile Hardware and Software Systems Engineering For Critical Military Space Applications*, in: *Sensors and Systems for Space Applications V*, Volume 8385 of *Proceedings SPIE 8385*, SPIE Digital Library, p. 1–9.

Kaisti M., Mujunen T., Mäkilä T., Rantala V., Lehtonen T., 2014: *Agile Principles in the Embedded*

System Development., in: *XP*, Volume 179 of *Lecture Notes in Business Information Processing*, Springer, p. 16–31.

Kaisti M., Rantala V., Mujunen T., Hyrynsalmi S., Könnölä K., Mäkilä T., Lehtonen T., 2013: *Agile methods for embedded systems development - a literature review and a mapping study.*, in: EURASIP Journal on Embedded Systems, 2013, p. 1–16.

Manhart P., Schneider K., 2004: *Breaking the Ice for Agile Development of Embedded Software: An Industry Experience Report.*, in: *Proceedings of the 26th International Conference on software engineering*, ICSE '04, IEEE, p. 378–386.

Meyer B., 2014: *Agile: the Good, the Hype and the Ugly*, Springer, 1 Edition.

Palmer S.R., Felsing J.M., 2002: *A Practical Guide to Feature-Driven Development*, Prentice Hall PTR, 1 Edition.

Reynisdottir T., 2013: *Scrum in Mechanical Product Development Case Study of a Mechanical Product Development Team using Scrum*, Thesis. Accessed: 20.11.2017. `http://publications.lib.chalmers.se/records/fulltext/191951/191951.pdf`

Sandhaus G., Berg B., Knott P., 2014: *Hybride Softwareentwicklung: Das Beste aus klassischen und agilen Methoden in einem Modell vereint*, Springer, 1 Edition.

Schindler C., 2010: *Review of Agile Software Development Methods in Practice*, Dissertation. Accessed: 12.12.2017. `http://diglib.tugraz.at/download.php?id=576a7a82525c9&location=browse`

Schuh G., Schroder S., Lau F., Wetterney T., 2016: *Next generation hardware development: Requirements and configuration options for the organization of procurement activities in the context of Agile new Product Development*, in: *2016 Portland International Conference on Management of Engineering and Technology (PICMET)*, Portland International Conference on Management of Engineering and Technology, Inc., p. 2583–2591.

Schuh P., 2005: *Integrating agile development in the real world*, Charles River Media, 1 Edition.

Schwaber K., Sutherland J., 2017: *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game*. Accessed: 28.07.2017. `https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf`

Shen M., Yang W., Rong G., Shao D., 2012: *Applying agile methods to embedded software development: A systematic review*, in: *Proceedings of the Second International Workshop on Software Engineering for Embedded Systems*, SEES '12, IEEE, p. 30–36.

Thompson K., 2015: *Agile Processes for Hardware Development*. Accessed: 8.09.2017.

`https://www.cprime.com/wp-content/uploads/woocommerce_uploads/2015/10/cPrime-Agile-Processes-for-Hardware-Development.pdf`

Wirdemann R., 2011: *Scrum mit User Stories*, Hanser, 2 Edition.

Appendix

Interviewleitfaden Masterarbeit

Fragen zur Personen

1) Was ist Ihre Rolle bei der AVL?
2) Wie viel Erfahrung besitzen Sie in Ihrer Rolle?
3) Arbeiten Sie an mehreren Projekten gleichzeitig?

Fragen zu Xion

4) Wie lange arbeiten Sie schon an dem Xion Projekt?
5) Was für eine Entwicklungsmethode wird derzeit in Ihrem Arbeitsbereich verwendet?
6) Welche Stärken hat diese Vorgehensweise?
7) Sehen sie auch Grenzen/Schwächen in der Vorgehensweise?
8) Was funktioniert gut bei der Entwicklung von Xion?
9) Was hätte man besser machen können?
10) Wie wird derzeit getestet(Simulationen, Hardware Tests, Unit Tests)?
11) Wie arbeiten die Abteilungen miteinander(Meetings, Jira,...)?

Fragen zur Agilen Entwicklung

12) Haben sie Erfahrung mit Agilen Entwicklungsmethoden?(In wie vielen Projekten haben sie schon agile Entwicklung eingesetzt?
13) Was würde sich verändern wenn agile Entwicklungsmethoden eingesetzt werden?
14) Wo sehen sie Stärken in der Agilen Vorgehensweise?
15) Wo sehen Sie Herausforderungen?
16) Könnten Agile Entwicklungsmethoden dem Projekt helfen? Wenn ja wie, falls nicht wieso?
17) Wie lange würde eine ihrer Meinung nach eine Iteration dauern und wie könnte man Firmware/Hardware/Software parallelisieren?
18) Herunter brechen von Anforderungen in Tasks ist ein Hauptbestandteil in den meisten Agilen Entwicklungsmethoden, könnte dies auch bei Xion funktionieren(Probleme, Vorteile)?
19) Wie könnte man die Probleme die durch die Abhängigkeiten von Hardware und Firmware entstehen lösen?

D: Gut beginnen wir. Ich sitze hier mit Herrn „interviewte Person“, einem Softwareentwickler. Bitte, „interviewte Person“, skizziere dein Aufgabengebiet bei der Firma AVL.

Interviewte Person : Ich bin als Softwareentwickler angestellt und bin Teil des Softwareteams von Indicom/MIE. Meine Hauptaufgaben ist die Softwareentwicklung. In der Software gibt es mehrere Bereiche, genauer gesagt Themenbereiche. Mein Hauptaugenmerk liegt auf der Umstellung von Indicom 2.8 auf indicom 3.0. Dies ist notwendig, da sich unsere Basisplattform des Concerto v4.8 auf v5.0 sehr verändert hat. Nun gilt es für uns die neue Plattform zu implementieren und dies gilt auch für die Indicom Unterstützung. Einige Features des Indicom 2.8 wie z.B. Booma Kopplung oder verschiedene Recorder Geräte mit Xion, unterliegen nicht meiner Zuständigkeit Dennoch musste ich mich mit mit Xion-Hardware auseinandersetzen – aktuell jedoch nicht mehr.

D: Welche Berufserfahrung als Softwareentwickler hast du bereits gesammelt?

Interviewte Person: In Bezug auf Firmenprojekte bin ich noch unerfahren. Die aktuelle Anstellung ist meine erste größere Firma, mit der ich einen längerfristigen Vertrag habe. Ansonsten hatte ich mehrere Ferialpraktika und Nebenjobs in dem Bereich.

D: Resümierend lässt sich festhalten, dass du Junior mit unter 3 Jahren Erfahrung bist. Welche Software Entwicklungsmethoden hast du bereits verwendet? Zwischenfrage:Wird im derzeitigen Projekt eine Entwicklungsmethode verwendet?

Interviewte Person: Es gibt innerhalb des Teams eine Methodik, zu der ich keinen Begriff in Erinnerung habe. Jedoch ist diese nicht sehr strukturiert. Weiters ist festzuhalten, dass ich bei sehr vielen Meetings nicht dabei bin. Dies ist damit begründet, dass ich mit XION nicht allzu viel zu tun habe.

D: Welche Schritte wurden von deinem Vorgesetzten vorgegeben?

Interviewte Person: Der Ablauf ist folgendermaßen: Entweder es erfolgt eine Themenzuteilung oder der Vorgesetzte weist die Themen zu. Im Großraumbüro erfolgen Diskussionen und es gibt den Defekt Prozess. Wird ein Tester oder ein Software Entwickler auf einen Fehler aufmerksam, so wird dieser eingetragen und gefixed. Dieses System basiert auf der Software Jira.

D: Welche Stärken hat diese Vorgehensweise?

Interviewte Person: Meiner Meinung nach: Nicht viele. Ich persönlich empfinde es als unstrukturiert und es ist nicht wie bei den Indicom Softwareentwickler. Diese haben mehr Meetings, koordinieren sich mit den Xion Firmwareentwickler, Hardware Entwicklern und anderen. Meine Schnittstelle zu den Indicom Entwickler ist zwar vorhanden, jedoch fallen diese ebenfalls unstrukturiert.

D: Kurz zusammengefasst: Der gesamte Projektablauf ist deiner Meinung nach unstrukturiert?

Interviewte Person: Ja.

D: Ist ein Großraumbüro eine Stärke in Bezug auf die Kommunikation?

Interviewte Person: Ja, ist es.

D: Die Kommunikationswege sind kürzer?

Interviewte Person: Ja! Wenn eine Frage aufkommt, wird diese sofort beantwortet. Das ist praktikabel.

D: Welche Testvorgaben gibt es?

Sample Interview

Interviewte Person: Was wird unter Testvorgaben verstanden?

D: Wie wird getestet?

Interviewte Person: Es gibt ein eigenes Testingteam, mit dem definiert wird, was getestet werden soll und welche Funktionalität gegeben sein muss. Jedoch kommt es oftmals zu problematischen Konstellationen, dass ein Hotfix freigegeben werden sollte, jedoch ein großer Fehler unerwartet auftrat , welcher nicht getestet wurde. Das bedeutet in weiterer Folge Stress.

D: Okay.

Interviewte Person: Es wird meiner Meinung nach zu lange zugewartet. Testen sollte viel früher passieren.

D: okay.

Interviewte Person: Es gibt auch andere Schwächen. Dadurch, dass ich nicht bei allen Meetings eingeladen bin, kann ich beim Thema Xion wenig beitragen. Mit Scrum wird sich etwas ändern.

D: Gut, das heißt in der Methode. Was würdest du verändern, weil du gesagt hast, dass es keinen fixen Prozess gibt. Es werde eher versucht, anstatt einem Prozess zu folgen.

Interviewte Person: Grundsätzlich gibt es den Prozess mit den User-Requirements. Dieser wird jedoch auf einer zu abstrakten Ebene praktiziert. Es gibt wenig Detailplanung und vor allem keine, die alle involviert.

D: Das heißt: Es gibt wenige, die das Wissen dazu haben?

Interviewte Person: Es ist Inselwissen vorhanden, jedoch ist das große Ganze nicht für jeden klar definiert. Das soll heißen, dass die Planung zu Detailarm ist, wenn es um Featuretesting geht, was getestet werden soll und ähnlichem.

D: Es gibt keine Exitdefiniton oder eine „Definition of done“, wie es im Scrum genannt wird?

Interviewte Person: Diese ist zwar vorhanden, jedoch meiner Meinung nach nicht funktionierend. Ansonsten würde das Zeitmanagement besser funktionieren.

D: Was müsste sich somit ändern?

Interviewte Person: Zu aller erst müsste es bereits während der Entwicklung Tests geben und nicht erst beim Abschluss. Somit könnten Abnahmen früher abgehalten und Features als „abgeschlossen“ markiert werden.

D: Wie wird getestet? Mit Unit Test oder händisch?

Interviewte Person: Das entzieht sich meiner Kenntnis.

D: Wenn du einen Code implementierst, schreibst du auch Tests dafür oder probierst, du ob der Code funktioniert?

Interviewte Person: Ich teste so gut ich kann, jedoch für detaillierte Tests wird der Code und das Testteam weiter gegeben. Bevor ich den Test in Jira freigebe, probiere ich grundsätzlich, ob alles funktioniert.

D: Ist dem Testteam bekannt, was die Software machen soll?

Interviewte Person: Das Testteam kennt die Aufgabenstellung und wissen, wie es beim Kunden eingesetzt wird und was wichtig ist. Sie wissen zum Beispiel, ob ein Kunde das Feature verwendet oder nicht. Ich habe dahingehend keine Ahnung; wäre jedoch wünschenswert.

D: Arbeitest du mit anderen Abteilungen zusammen?

Interviewte Person: Ja, mit der Plattform Concerto, Team classic und Team noise. Wir stellen wie erwähnt auf die neue Plattform um. Indicom und Concerto waren eine Software bis v2.8. Das wird nun aufgebrochen, nachdem zwei Executables veröffentlicht wurden. Concerto in der Version 5 beinhaltet noch viele Teile von Indicom und es besteht eine enge Zusammenarbeit, um die Teile zu separieren. Der Löwenanteil fällt dabei auf mich ab, weil ich mehr Zeit dafür aufbringen kann. Die anderen beschäftigen sich hauptsächlich mit Features der Verson 2.8 und dem Bugfixen.

D: Findet die Kommunikation persönlich statt oder wird diese über das Jira abgehalten?

Interviewte Person: Tritt jemand des Concerto-Teams an uns heran oder diese implementieren ein neues Feature das mit dem Indicom negativ interferiert, dann versuchen sie es mit einem Fix des Indicoms und erstellen ein Git pull request. Diesen Request begutachten wir und akzeptieren, was übernommen werden soll. Wurden Probleme festgestellt oder notwendige Features entfernt, so wird ein Meeting anberaumt. Der Alaska Prozess ist jedoch noch nicht umgesetzt. Dadurch soll alles strukturierter ablaufen.

D: Weil du den Alaksa Prozess angesprochen hast: Hast du bereits Erfahrung mit agilen Entwicklungsmethoden? Hast du es schon verwendet?

Interviewte Person: Wie bereits erwähnt ist das die erste Firma, bei der ich längerfristig angestellt bin. Deswegen habe ich dementsprechend keine Erfahrung.

D: Das heißt, du musst auf theoretisches Wissen zurückgreifen?

Interviewte Person: Theoretisches Wissen ist vorhanden. In kleinen Teams an der Universität oder HTL wurde es ebenfalls eingesetzt. Grundsätzliches Verständnis dafür ist vorhanden.

D: Was würde geschehen, wenn in deiner Abteilung eine agile Entwicklungsmethode eingeführt wird? Wird sich etwas verändern?

Interviewte Person: Ich hoffe, dass es eine Struktur in den Prozess bringen wird. Alaska fordert beispielsweise Meetings mit einer Dauer von 4 Stunden alle 2 Wochen; Sprint planning im Fachjargon. In diesem wird geplant, was zu tun ist, wer für was zuständig ist und gemäß der Definition of done vorgegangen. Durch die Einbindung mehrer Leute und das Vorbringen Ihrer Bedenken sollten sich positive Aspekte ableiten lassen.

D: Du stehst der agilen Entwicklung positiv gegenüber?

Interviewte Person: Ja.

D: Gibt es in den derzeitigen Meetings einen externen Scrum-Master, der die Einschulungen übernimmt?

Interviewte Person: Das Kick-off Meeting findet am Montag statt und wird durch einen externen Scrum-Master begeleitet. Die Einschulung wird während zwei oder vier Sprints stattfinden.

D: Wie stehen deine Kollegen dem Thema Alaska gegenüber? Wird von oben diktiert oder ist es der Wunsch aus eurem Team?

Interviewte Person: Ich bin in den Entscheidungen der oberen Ebenen nicht involviert, jedoch wird alles dort beschlossen, jedoch stehen dem alle überwiegend positiv gegenüber. Es gab selbstverständlich gegebenüber detaillierten Themen Bedenken, weil das Scrum Team Software, Hardware und Firmware beinhaltet, jedoch wird es sich weisen, ob diese berechtigt sind.

D: Betreffend der Hard-, Firm- und Software: Wie wird die Zusammenarbeit mit den Teams ablaufen? Werden die Abstimmungen räumlich getrennt oder gemeinsam am Tisch erfolgen?

Interviewte Person: Nein, darüber wurde noch nicht gesprochen. Das Meeting bezüglich Agilität findet kommenden Montag statt. Aber es sitzen bestimmt nicht alle im selben Raum.

D: Meinst du, dass dein Team näher mit dem Hardware- und Firmware-Team zusammenarbeiten wird müssen, um neue Software entwickeln zu können oder wird sich nichts ändern?

Interviewte Person: Darüber kann ich keine Auskunft geben, denn ich setze mich derzeit kaum mit Hardware und Firmware auseinander.

D: Dann wechseln wir das Thema und fokussieren uns auf Sprints Hat jeder im Team Sprints? Wie werden diese aligned oder synchronisiert? Wie sehen die Abstimmungen mit den Firm-, Hard- und Software aus? Die Software wird bereits mit der Hardware abgestimmt, und dergleichen. Wie kommt es zu inkrementellen Releases?

Interviewte Person: Das war eines jener Themen, die beim Scrum Meeting angesprochen wurden, aber es kann selbst definiert werden, was ein inkrementelles Realease ist. Was sollte ein laufendes Produkt haben? Das ist beim Thema Hardware etwas schwierig. Während des Scrum Trainings werden verschiedene Ansätze angesprochen. Es obliegt jedem selbst, zu definieren, was inkrementell ist, beispielsweise Simulationen. Ich kann dir bei „alignen“ nicht folgen. Meinst du Sprints synchronisieren? Soweit ich weiß wären wir nur ein Scrum Team sein. Das heißt, wir haben alle die gleiche Sprint-Dauer?

D: Es existieren im Team nicht verschiedene Prozesse? Es wird doch Alaska und Aladin verwendet: das sind zwei verschiedene Prozesse: einer für die Hardware und einer für die Software.

Interviewte Person: Ich vermute, wir sind nicht nur im Aladin. Wir sollten die selbe Sprint-Dauer haben, jedoch habe ich keine offiziellen Informationen darüber. Dies wird von den Verantwortlichen besprochen. Diese Frage wurde jedoch beim Scrum-Training aufgeworfen: Ist die Dauer eines Software-Sprints genauso lange wie ein Hardware-Sprint mit zwei Wochen definiert? Der Fortschritt bei der Hardware wird in 2 Wochen wohl weniger eklatant ausfallen. Wir sind bereits gespannt auf die Umsetzung.

D: Tasks runter brechen wird ebenfalls eingesetzt, richtig? Es existieren mehrere Stories oder Requierements und wie läuft das bei euch ab? Gibt es eine bestimmte Vorgehensweise, wie „runter gebrochen“ wird? Oder ist das von der Erfahrung des Entwicklers abhängig?

Interviewte Person: Das weiß ich eigentlich nicht, ich bekomme grundsätzlich meine Tasks vom Vorgesetzten zugewiesen. Story wird prinzipiell nicht in Jira in Tasks unterteilt. Wir haben keine Tasks sondern Stories, die wir abarbeiten.

D: Wird sich das durch die Einführung von Scrum ändern? In Scrum wird in Requierements unterteilt.

Interviewte Person: Ich denke, dass sich die Abläufe ändern werden. Jedoch mit Sprint planning werden die Stories feiner in Arbeitsaufträge unterteilt.

D: Wird es Probleme geben, wenn keine erfahrene Entwickler im Team sind, die die Aufteilung übernehmen?

Interviewte Person: So ein Fall wäre natürlich nachteilig, jedoch gibt es genügend erfahrene Entwickler, somit wird dieser eher nicht eintreten. Zumindest hoffe ich das.

D: Denkst du das es sich Agilität positiv auf die Software Entwicklung und dadurch auch auf Indicom auswirkt?

Interviewte Person: Also ich denke das es sich auf jedenfall positiv auswirken wird, jedoch wird es eine Zeit lang dauern.

D: Kannst du mehr über Agilität berichten? Welche Hoffnungen hast du? Wo kann es zu Problemen kommen? Wie praktikabel ist Agilität?

Interviewte Person: Ich persönlich ist es sinnfrei, dass im Scrum die sich die gesamte Kompetenzen in einem Team vereinen sollt. Somit sind alle für alle Bereiche zuständig. Umgesetzt auf meine Situation hieße es, dass ich gemäß der Definition von Scrum auch Hardware entwickeln müsste, obwohl mir das Wissen dazu fehlt.

D: Ist es nicht eher so, dass es verschiedene Teams gibt, wie in diesem Beispiel jeweils eines für Hardware und für Software und jedes für sich einen Scrum-Master hat?

Interviewte Person: Das ist eben einer der offenen Punkte. Im Scrum-Training wurde uns mitgeteilt, dass ein Team alle Kompetenzen vereinen soll. In unserem Fall wäre das bei Indicom eben Hardware und Software, weil das als ein Produkt gesehen wird, das entwickelt werden soll. Dieses Scrum-Training war jedoch nicht spezifisch auf Aladin oder Alaska bezogen sondern behandelte Scrum im Generellen. Wie das in Aladin oder Alaska gehandhabt wird, weiß ich leider nicht. Wir sind auf alle Fälle ein gemeinsames Team mit einem Scrum-Master, content owner, produkt owner und so weiter. Das ist mein derzeitiger Informationsstand.

D: Erfahrungsgemäß würde ein Sprint in der Softwareentwicklung wie lange dauern?

Interviewte Person: Meines Wissens sind zwei Woche angedacht.

D: Ist eine Iteration ausreichend um als ein funktionierendes Inkrement zu gelten?

Interviewte Person: Mit meinem heutigen Wissensstand kann ich diese Frage nicht ausreichend beantworten. Von Seiten der Software sehe ich es im Vergleich zur Hardware oder Firmware weniger problematisch. Jedoch fehlt mir in den beiden zuletzt genannten Teilen die Erfahrung.

D: Im Scrum ist ein kontinuierlicher Testablauf vorgesehen. Wird dieser in Zukunft durchgeführt werden oder wird sich nichts ändern?

Interviewte Person: Es wird bereits kontinuierlich getestet.

D: Du hattest erwähnt, dass Fehler erst am Ende entdeckt werden.

Interviewte Person: Ja, das stellt ein Problem dar. Größere Bugs werden erst am Ende entdeckt. In wie weit sich das in Zukunft ändern wird, weiß ich nicht. Es fehlt mir auch die Kenntnis darüber, wie die Fehler passiert sind: Kümmerte sich niemand darum? Sind sie im Laufe der Entwicklung entstanden? Scrum wird diese Problematik meiner Meinung nach nicht lösen.

D: Wurde im Scrum-Meeting darauf eingegangen, wie die Entwicklung der Hard-. Soft- und Firmware parallelisiert werden kann?

Interviewte Person: Nein, eben nicht. Es zielte nicht speziell auf Alaska und Aladin ab, sondern es war ein Überblick über Scrum. Synchronisation kam dabei nicht zur Sprache. Das lag auch daran, dass der Trainer keinerlei Erfahrung mit Alaska und Aladin vorweisen konnte, sondern nur mit Scrum vertraut war. Es wurden 3 Teams (Hardware, Firmware und Software) vorgeschlagen, die sich über einen Scrum of Scrum synchronisieren sollen. Wie das jedoch im Endeffekt ausschauen wird, ist für mich unklar, da wir, wie bereits erwähnt, ein Team sein sollen.

D: Wenn ich das richtig verstanden habe, wird nicht Komponente für Komponente entwickelt? Es gibt also kein Komponenten-slicing?

Interviewte Person: Ich denke, die Komponente in meinem Fall ist Indicom. Es existiert aber auch diie SIU. Dieses Gerät benötigt selbst eine Software. Der Entwickler von SIU implementiert es für Indicom. Näheres würdest du bei unserem SIU-Entwickler erfahren.

Angestellten Interviews

Allgemein							
Rolle der bei der AVL	Elektronik Entwickler für Tests; Projektleiter/Program manager über mehrere Projekte; Hardware Entwicklung Indizier Messtechnik; Firmware/Hardware Entwickler; Softwareentwickler; Projektleiter; Abteilungsleiter						
Erfahrung	5 - 30 Jahre						
Aufgaben	Prüfvorschriften schreiben; Hardware/Firmware/Software/FPGA entwickeln; Hardware design; Oberflächen Entwicklung						
Fragen	**Interview 1**	**Interview 2**	**Interview 3**	**Interview 4**	**Interview 5**	**Interview 6**	**Interview 7**
Arbeiten Sie an mehreren Projekten gleichzeitig?	Xion und andere	Xion; Betreuung von Altprojekten	Xion und andere Baustellen;	Projekte von Mitarbeiter der in Pension gegangen ist	Schnittstelle zu den Verstärkern; zentrale Anlaufstelle für	Entwickler für Indicom	mehrere Projekte für schnelle Messtechnik; teilweiße

Angestellten Interviews

				hat er ein paar bekommen, eher Serien Betreuung, hauptsächlich aber Xion	Fehler in Indicom		noch in der Entwicklung; Indicom Setup
Herausforderungen in der HW Entwicklung?	Zeitproblem, muss schnell gehen; was ist das richtige Bauteil (kommt durch Erfahrung)	Lange Wartezeiten beim bestellen	Lieferzeiten; Durchlaufzeiten; Reale Objekte				Lieferzeiten; testen
Herausforderungen in der FW Entwicklung	Focus bei mehreren Projekten; selbst beigebracht; testen wenn kein Prototyp da ist	Entwickeln ohne Hardware		selbst Firmware entwickeln gelernt			Entwickeln ohne Hardware
Wie lange arbeiten Sie schon am Xion/Projekt?	Seit Anfang	Von Anfang an dabei	von Anfang an dabei	>5 Jahre	schon seit Begin	erst seit 3 Jahren ist Indicom bei Xion dabei; entwickelt wird Parametrisierung, Ansteuerung für die Hardware über einen Softwaretreiber zur Datenerfassung n	konkret nicht in der Entwicklung; nur mitreden um Ideen und Erfahrungen einzubringen; erstellen technischer Requirements

Angestellten Interviews

Was für Entwicklungsmethoden wird derzeit verwendet?	Wasserfallmodel; mit Rückschritten zu vorige Gates; Pflichtenheft, Lastenheft, grober plan, Layout, Konstruktion, Evaluierungsarbeiten, testen; Konzepte erarbeiten; Kommunikation mit Kollegen	PIP, dann neuer PIP, wird kaum richtig gelebt	Datenanalysen; kein komplett neues System; schon bekannt was zu machen war; Machbarkeitsstudie, Demo Projekte gemacht, Layout Simulationen(Siemens) trotz Preis; Standard Geräte Entwicklung aber keine echten Kunden Projekte	Evaluationboard in Betrieb nehmen; sequentieller Ablauf entwickeln; Projektleiter, (Keine Vorschläge) definiert was zu tun ist	Wasserfall massige Entwicklung; mim 1-2 entwickeln, sehr großer Aufwand aber es ist gut gelaufen; iterativer strukturierter; durch Xion Problem, weil neu entwickeln; war langsam am Anfang, sehr chaotisch; unkoordiniert; alles so wie damals machen und deswegen waren die Mitarbeiter nicht zufrieden	PIP; seit 25 Jahren, sehr Hardware lästig; CMMI Prozess für Software eingeführt worden; wird aber nicht mehr so gelebt; Wasserfall Model, wenig Komplikationen zwischen Hardware, Software; seit 4 Jahren neuen PIP; basiert auf Stationen jetzt ist er sehr restriktiv geht nicht das man in eine nächste Station gehen, wenn kein go von	PIP wird derzeit für neue Geräte eingesetzt; Overhead für kleine Produkte ist zu groß (XION in anderes Gehäuse setzen); PIP beschreibt mehrere Stufen

Angestellten Interviews

						Stakeholder da ist	
Welche Stärken hat diese Methode?	man weiß was zu tun ist; kaum Problem bei der Entwicklung	kaum stärken; vieles macht Sinn aber es ist alles zu viel	Kommunikation war super Externe Mitarbeiter sind experten, gute Allrounder aber keine Spezialisten	sehr detailliert geplant, leichtes wechseln zwischen Projekten	Feature nach Feature entwickeln	Dokumentation zum Teil gut; wenig Leute haben damit zu tun gehabt; strukturierter gearbeitet	es ist sehr strukturiert; für komplexe Geräte ist es teilweise gut, solang direkt mit den Personen geredet wird
Sehen sie Schwächen/Grenzen in der Methodik?	Warten auf andere Abteilungen; kaum Entscheidungen gefallen;	Ressourcen Probleme; Stolpersteine; wenig Führung; Konsens zwischen Management fehlt; alles muss schon erarbeitet sein (Konzepte, Tests, Prüfkonzepte); sehr starr	viele Verzögerungen aber keine Fehlentscheidungen, Management hat nicht auf dich gehört, Ressourcen Probleme durch beschränkte Kapazitäten, Sparmaßnahmen , Durchlaufzeiten; Entscheidungsfre	Fokus verloren, Ressourcen haben gefehlt; lange Bestellzetteln ; Änderungen schwer bei Prototypen zu machen; externes simulieren(Siemens)	nicht koordiniert, Caos, sehr unrealistische Termine,falsch geplant; Termine waren immer fix trotz andere höher priorisierter aufgaben	starr; alle Dokumente müssen da sein die man braucht; viel planen; neuer PIP erschwert Entwicklung ohne go von Stakeholder geht nichts weiter; großes ganze hat gefehlt;	sehr großen Overhead; zu viel planen: man muss warten bis alle Stakeholder ein "Go" geben um in die nächste Phase zu gehen, wird oft blockiert um keine Verantwortung zu

			iheit begrenzt, Verantwortung wurde weiterverschoben			jeder entwickelt wir er glaubt	übernehmen; nicht parallelisierbar, sehr demotivierend, trotz vieler Reviews kommt es beim Produkt zu Problemen; kein gemeinsamer Fokus; Projektleiter muss alles selber machen
Was hätte man besser machen können?		nicht auf alles warten; nicht ganz an dem Prozess halten	mehr Unterstützung/Verständnis seitens des Managements; früher auf Probleme reagieren		aber es nicht koordiniert und dass wollen wir ändern, jeder weiß was der andere macht, und es besser kommuniziert wird was gemacht wird, Test wird besser informiert, wenn man schrumm einführt, wir		PIP auflockern (für kleinere Projekte); Personen nur miteinbeziehen, wenn man sie auch braucht; Vorausschau was für Bauteile benötigt werden

Angestellten Interviews

					haben zu wenig Ressourcen gehabt für dass was zu tun war, Abschätzung war dafür so und so viel Leute aber die Leute haben gefehlt, und auch keine Zeit dafür gehabt		
Wie testet man derzeit?	Serientests; Funktionstests; Leiterplatten beim Hersteller; Fertigungsendtest; immer an der Hardware; Evaluirungsboards	Testen durch Simulationen; an Prototypen, während Entwicklung	Layout Simulation; Thermische Simulation; FPGAs Schaltungen simuliert	an Evaluationboard; Feinheiten testen nur am Prototypen; Layout Simulationen ; Test Funktionen für Firmware; VHDL Code testen; später in Indicom einbinden	getestet wird in der Test Abteilung; Unit Checks; automatischer Test Sever; high Level Tests		jeder für sich; später dann zusammen; automatische Tests; Unit Tests

Angestellten Interviews

Wie arbeiten die Abteilungen miteinander (Hardware, Software, Firmware)?	ungefähr wissen was jeder macht und an wenn man sich wenden kann/ Face to Face reden	wöchentliche Meetings; Face to Face	nicht ganz Sequenziell; bisschen parallelisiert; anhand (Keine Vorschläge) Firmware entwickeln; eher Interrupts von der Softwareseite	direkter Kontakt bei Problemen oder wenn was besprochen wird, oder gemeinsam durchschaue n, Meetings (Statusmeeti ngs HW, Indicom)	von drüber reden was zu tun ist bis, Hardware hat schon eingebaut und Software muss es jetzt auf ihrer Seite einbauen, hier gibt es großes Verbesserungs potenzial, weil das Verständnis fehlt wie einzubauen ist, es wurde parallel entwickelt, aber bis Xion da war haben wir schon viel gebaut gehabt, Person hat gefehlt die sich für Software und Hardware kümmert die schaut das bei beidem was gebaut wird, es hat nur gut funktioniert weil jeder die	Software erst mit den Xion prototype dabei: Software hat sich auf Firmware ausgeweitet , Kommunikat ion nicht passt, Focus war anders, gleiche Abteilung 2 Teams irgendwie zusammeng erauft, Software wurde zu spät eingebunde n	müssen sich zusammenra ufen; Personen werden miteinbezoge n wenn gebraucht

					Leute kennt und ein gutes Verhältnis gibt, Kommunikation war nicht immer gut, zusammenführen immer dann wenn sich die Kollegen gemeldet haben		
Kann man modular arbeiten?	Modularität geht nicht; in Prinzip wird's gemacht, aber nicht von Anfang an	ja kann man, tun wir schon zum teil	Xion ist modular; dafür aber teuer;				Ja könnte man, ist aber nicht so leicht umzusetzen
Wie lang dauert es bis ein Prototyp da ist?	mindestens. 2 Monate; Bauteile werden vorab bestellt ca. 8 Wochen Lieferzeiten; wenn Fehler am Prototyp dauert es;	ca. 3 Monate	Halbes Jahr bis es beim Kunden ankommt	2-3 Monate		Dauert sehr lange, Fehler würden zu langen Wartezeiten führen, Simulationen würden da helfen	hängt von den Lieferzeiten von Bauteilen, vom Mechanik Produzenten, von den externen Lieferanten ab; kann mehrere

Angestellten Interviews

							Monate dauern
Was macht ihr an den Übergangszeiten (bis Prototypen da sind)?	an anderen Projekten arbeiten	an anderen Baustellen arbeiten	an anderen Projekten weiterarbeiten	an anderer Firmware arbeiten			es werden an anderen Baustellen weiter gearbeitet
Haben sie Erfahrung mit Agilen Entwicklungsmethoden?	Keine Erfahrung; keine Schulung; durch Kollegen erfahren wie es funktioniert	Schulung; im Prinzip sind wir schon agile	Keine praktische Erfahrung; weiß aber grob worum es geht	Schulung gehabt; erster Kontakt zur agilen Entwicklung	Einschulung gehabt	ja bisschen durch ALASKA, Einschulung	Keine praktische Erfahrung, war bei der Schulung dabei
Was für Veränderungen würden eintreten, wenn agile Methoden eingesetzt werden?	mehr Reportet;Team weiß mehr was genau passiert	kann er nicht genau sagen	kennt die Methodik nicht in der Hardware; richtiges abschließen von Projekten; bessere Koordination	Agilität in der Firmware geht, nach Features entwickeln; in Sprints entwickeln; daily standup ist super da man dann von jedem weiß was er macht	Scrum nur für dinge die Indicom betreffen z.B. Firmware und teilweise Hardware für Indicom Funktion aber nicht Gehäuse oder Spannungsversorgung; viele Meetings mit viel Aufwand am Anfang	wenig druck und bessere Zusammenarbeit; Focus mehr auf Funktionalität Mindset; sehr viel Eigeninitiative, Firmenstruktur würde sich ändern (Teamleiter, Linienleiter), Management muss sich	starke Verantwortung für die Gesamtheit des Produktes; Hardware Team wird miteinbezogen; Feature orientierte Entwicklung

Angestellten Interviews

						daran auch gewönnen	
Wo sehen Sie stärken in der Agilen vorgehensweiße?	Focus an Features; mit Team abstimmen; sehr flexibel; Informationsaustausch	Kommunikation; besseren Focus; kein Einmischen; fixes Team; fixes Ziel	abarbeiten der Problemstellungen in 2 Wochen; Priorisierungen; Kommunikation wird sicher besser; co Location ist schon vorhanden	richtiges abschließen von Features mit testen; weniger Projekte gleichzeitig	bessere Koordination, besseres Planung, bessere Sichtbarkeit, neu Priorisierungen	bessere Kommunikation, besser verstehen, strukturierter, transparent was jeder tut, Knowledge sharing, bessere Aufteilung weniger Spezialisten, bessere Architektur, bessere Entscheidungen zusammentreffen, Prüfungen die im PIP sind könne durch agile Vorgangsweißen	koordinierteres arbeiten; bessere Kommunikation; Verantwortung geht über ans Team; besserer Fokus

Angestellten Interviews

						ausgelassen werden	
Wo sehen Sie Schwächen/Grenzen?	2 Wochen Sprints sind kurz für Hardware, eher Softwarebezogen; 2 Monate bis Prototypen, bis Schaltplan, Layout (ca. 2 Monat) je nachdem wie komplex; Simulationen; Inbetriebnahme; viele Meetings	Hardware Team muss alles können; Ressourcen fehlen; mehrere Backlog für verschiedene Projekte; ein großes Standup Meeting; Planung der Teams; Lieferzeiten ändern sich nicht; Weisungsgebundenheit; Kultur Change	Lieferzeiten; Beschaffungszeiten; Verzögerungen durch Testzyklen auch bei Simulation; Hardware schwer in Sprints planbar, Planung der Mitarbeiter für nur ein Projekt	große Abhängigkeiten durch die Hardware, bestellen geht nicht agile; review/retrospektive braucht viel Zeit	schwer vorzustellen wie das mit der Koordination mit der Hardware ist, gespannt wie es beim Release wird, rückfallen auf alte (Keine Vorschläge)	zurückfallen auf alte Gewohnheiten bei Krisen Zeiten; Priorisierung von Projektleiter aufgaben	komplett agile Hardware entwickeln ist schwer; muss unterschieden werden zwischen programmierbarerer Hardware und reiner Hardware; Hardware sollte eigentlich fertig entwickelt werden, zusätzliche Funktionen werden über Indicom bereitgestellt

Angestellten Interviews

Wie könnten die Hardware, Software, Firmware zusammenarbeiten?	Durch mehr reden; Meetings	Firmware vorher an Demoboards entwickeln während Prototyp entwickelt wird; Kommunikation; ein Backlog	Bereiche arbeiten teils parallel; während Layout in der Bestellung ist, wird geschaut ob FPGA Pins passen, während dessen Firmware entwickeln; Firmware an Demoboards testen;	Firmware während der Entwicklung der Hardware programmieren auf low Level ebene; mit Evaluation boards; testboards; Evaluirungsboards verwenden um Abhängigkeiten zu lösen, Hardware derzeit nicht agile	treffen bei Meetings; diskutieren was zu machen ist, was geplant wird; ein Team: Firmware-, FPGA-, Hardware Entwickler, vielleicht eigenes ein Team, Hardware gibt Puls an	Ein Team aus Software und Firmware; Hardware kommt irgendwann dazu; wird sich von selbst einstellen; vielleicht zweites Team aus Hardware und Firmware; Produktmanager gibt an was zu tun ist; Hardware über programmierbare Bauteile; Hardware-/ Firmware requirements schon früh einbinden; Treiber ohne	Entweder als einziges Team oder mit mehreren Teams; ein gesamtes Backlog

Angestellten Interviews

						Hardware entwickeln, Simulationen, emulieren, auch umgekehrt möglich; Hardware modularer entwickeln	
Wo gebe es Schnittstellen?		ein Backlog; Abnahmekriterien; enge Mitarbeit	durchs Backlog besser ersichtlich für jeden was zu tun			Anforderungen austauschen, gegenüber offen zeigen	Backlog, Meetings
Was für Herausforderungen gäbe es da?	Firmware derzeit nach Hardware; deswegen schwer zu parallelisieren	Verzögerungen sind teils schwer vorherzusehen; Lieferanten; Physikalische Einflüsse auf Hardware	Kundenwünsche die auf einmal auftauchen	quick and dirty entwickeln geht nicht für komplexe Sachen	neue Hardware braucht mehr als einen Sprint, entwickeln nur auf bestehender Hardware; keine komplette Hardware Entwicklung	Unwissenheit; Missverständnisse warum jemand was (nicht) tut, falscher Fokus	Sprint länge; Definition vom Inkrement; Modular Hardware entwickeln für alle Arten von Feature; Grund Gerüst zur Verfügung stellen; Hardware produzieren und liefern auch ohne Software,

Angestellten Interviews

							Hardware Team nicht nur auf ein Produkt beziehen
Wie könnte man sie lösen?	Protoboards	Hybrid durch teil Agilität, Wasserfall für Hardware	kann man nicht ablehnen; Kundenwünsche sind wichtig; neu Priorisierung			besser informieren(Firma); ALADIN in Programm einbinden; vielleicht keine Unterscheidung zwischen ALASKA & ALADIN, Management Mindset anpassen	4 Wochen Sprints würden gehen (auch für Hardware); Task Definition könnte was anderes sein
Kann ein funktionsfähiges Inkrement rauskommen?	Schwer bei Hardware, Konzept geht	Layout; Bestellliste; Designs; Prototyp braucht länger	Planen, CAD, Leiterplattenpläne	bei der fertigen Hardware Firmware Features schnell entwickeln um ein Inkrement zu bekommen; Simulationen			Inkrement könnte auch Konzept; (Keine Vorschläge), Simulationen, FPGA Definitionen sein

Angestellten Interviews

				oder Pläne zeigen			
Herunterbrechen von Requirements?	Hängt von der Komplexität ab; meist nur als großes ganzes	Jain; quick and dirty entwickeln geht nicht; gekaufte Demo boards würden helfen; Firmware würde es funktionieren; Layout herunterbrechen möglich	Verlegung einer Leitung oder neue Buchse im System geht nicht als kleiner Task; ganzen bestell/Test Zyklus von neues durchlaufen; im Vorfeld URS und Listen Requirements fixieren;	Gemeinsames herunterbrechen; sehr zeitintensiv; eher ungenauer herunterbrechen	Ist möglich; wir haben Features in kleine Tasks heruntergebrochen; Langzeittask von Xion über nächsten Quartale müssen vorgeplant werden		wird schon gemacht, wird zusätzlich in Sprints aufgeteilt
Wie sind die Hoffnungen?	sieht es neutral; unvoreingenommen; mehr Transparenz	wird skeptisch gesehen, weil eher nur halb gemacht wird	man muss es einfach probieren um zu sehen, besser planbar, sehr positive Haltung solange man es real betrachtet	Overhead soll weniger werden; richtiger Abschluss von Features	Alle sind positiv eingestellt, ich bin in einer positiven Erwartungshaltung, kann mir, dass bisschen schwer vorstellen aber ich lasse mich überraschen, bessere koordinierte Feature Entwicklung, gemeinsam am gleichen		Produktmanagement muss zusammen entscheiden was mit welcher Priorität zu tun ist; Entscheidungen müssen getroffen werden

					arbeitet, gemeinsames besseres Verständnis wie viel geht bis zu einem gewissen Zeitpunkt in das Produkt, jeder sieht es und kann sich ein Bild machen, alle wissen was es ist und alle wissen was in 2 Wochen weitergangen ist und was noch fehlt		
Welcher Prozess wird verwendet(Aladin/Alaska)?		Keine Ahnung, Hardware ist derzeit nicht involviert worden	Ist noch nicht fixiert	Nur auf Scrum eingegangen	nur Alaska, bei Aladin gestartet aber dann zu Alaska, weil wir zu Hälfte aus Concerto bestehen welches Alaska verwendet	ALASKA, ALADIN später	ALASKA, ALADIN

	ALASKA	**ALADIN**
Idee	ALASKA ist kein PIP; einzelne Teams auf Scrum; Scrum allein reichte nicht, fehlte Framework zum Koordinieren von Teams, Safe	Klassische Hardware in die Agile Entwicklung; ein bisschen angelehnt an ALASKA; 4 Pilotprojekte zum Ausprobieren; Rahmenbedingungen in der Hardware anders als in Software, Adaptieren und Abänderungen des Frameworks anhand von Ideen
Aufbau	Portfolio Level als kontinuierlicher Prozess; neue Epic zur neuen Programm Iteration hinzugefügt; Programm Iterationen mit 4 Sprints, ein Innovationssprint, Retrospektive, Review auch auf Programm Level ebene; Programme sind nach ausgewählten Punkten ausgesucht	Indicom->Firmware-> Hardware; Agile + Wasserfall(PIP); nicht so streng sequenzielle Phasen; PIP als big Picture und Scrum als Motor welches Value erzeugt
Systemdemos/Demos	Programme bestehen aus mehrere Produkten; alle Produkte werden getestet	Demos müssen nicht unbedingt etwas physikalisch sein aber müssen ein Value haben; Testen wann es immer geht
Teambacklog	bestehen aus Tasks von mehreren Programmen	ein Backlog für das gesamte Team
Ablauf	PIP definiert Requirements und Abhängigkeiten; ALASKA entwickelt und der Markt Release geht über PIP; ALASKA als führender Prozess; Scrum of Scrum für Teams innerhalb eines Programmes (nur Scrum Master)	Scrum Basis Schulung; 2 Sprints mit Coach als Scrum Master danach eigener Scrum Master; Releaseplanmeeting, nach 2 Wochen einen Teilwert erschaffen der einen Wert kreiert mit Firmware, Software, Hardware; Feedback kommt schnell, Fehler werden schneller gefunden; Einkauf, Fertigung ist schon bei Prototypenbau dabei, gegen Ende immer mehr im Team beteiligt; erste Pilotteams
Features	Programm Features fein auf die jeweiligen Teams geteilt, dann auf Stories; Features können während des Sprints jederzeit in den Backlog aufgenommen werden	Kommen vom Produktmanagement; werden aufgebrochen in Tasks
Sprints	sind fix auf 2 Wochen gesetzt	2-4 Wochen

Unterschied in der Hardware		potential auslieferbares ist etwas das intern wert kreiert wo Feedback entsteht; Einkauf und Fertigung nicht Agile; Frühen Phase (Testaufbauten, Prototypen) mehr Agilität Rapidprototyping; vielleicht anderen Lieferanten wählen; alles wissen ist schwer, weil es zu große Unterschiede gibt aber es geht bis zu einem gewissen Grad

Best Practices Interviews

Best Practices/Allgemein	**Practice Interview 1**	**Practice Interview 1**
Bereich	Hardware Entwicklung	E-Storage System zum Testen von Batterien/E-Motoren; Software-Hardware Regelungen; 1 Basis Produkt wo Derivate gemacht werden
Unterschied zur Software	Viel größere Breite; Abhängigkeiten zu Lieferanten und anderen physikalischen Dingen	Abhängigkeiten zu Lieferanten, Reales Produkt
Anzahl der Lieferanten	2	
Abhängigkeit Lieferanten	Bestellungen dauern Ansicht nicht lange, nur Änderung dauern lange, meistens warten die Lieferanten auf Änderungsvorschläge	Lange Bestellzeiten
Prototypenablauf	Release Plan erstellt welches Abhängigkeiten berücksichtigt; funktioniert für komplexe Geräte; Funktionsmuster dauert ca. 3 1/2 Monate; es wird viel mit FPGA, Lochrasterplatinen gearbeitet	
Requirements:	Kommen von Produktmanagement; ebenfalls Änderungsvorschläge	Kommen von dem Produktmanagement und werden mit Produkt Owner besprochen
Framework	ALADIN als Framework; mehr Kundenzentriert; strebt Denk Änderungen an; Stories werden in Jira eingetragen und auf der Pinnwand aufgehängt; es wird Systemengineering für Requirements	Aladin Pilotprojekt; Produkt Owner stimmen sich unter einander ab, haben ein eigenes Meeting und eins mit dem Productmanagement; vor Selektion
Produkt Owner	Produkt owner als Projektleiter macht den Release plan und sagt was zu tun ist aber nicht wer es tun soll	3 Produkt Owner: 1 für das Produkt, 1 für Innovationsthemen, 1 für Kunden spezifische Varianten
Scrum Master	1 Scrum Master; Hütet das Framework; versucht eine starke Orientierung für das Ziel einzuführen	1 Scrum Master; kontrolliert ob das Scrum Framework eingehalten wird

Best Practices Interviews

Planen	Es wird auf bis zu 1 Jahr grob mit Milestones und Abhängigkeiten geplant und bis zu 2 Monaten detailliert; jeder Sprint hat ein Ziel; Scrum of Scrum wird verwendet; Backlog refinement wird über den gesamten Release plan aufgeteilt	Roadmap mit den Zielen; Sprint Planning 1: mit Feature Vertreter(wechseln je nach Aufgaben), Fertigung , ca. 1 Stunde , verstehen um was es geht; Sprint Planning 2: Mitarbeiter werden dazu geholt welches benötigt werden, kleinere Team stimmen sich untereinander ab; Daily Standup: alle Entwickler, ca. 20 Minuten; Sprint Review/Retrospektive: alle Entwickler, 1 1/2 Stunden Präsentation; Backlog Raffinement: Verantwortliche nehmen teil machen vor Selektionen vor jedem Sprint, 2 Stunden; Produktmanager Meeting: vor dem Backlog Refinement, Stories werden betrachtet
Backlog	Es wird ein Backlog verwendet; über 3-4 Sprints geplant (mit Einkäufer);	Jira für grobe Features (mehr als 1 Sprint); Stories sind abgeleitet von Features für 1 Team; 4 Scrum boards für die jeweiligen Produkt Owner; wird 3 Quartale vorrausgeschaut, feinere Planning für diesen Sprint; es wird in Done, Current, Next,3 Quartale aufgeteilt; spezielle Zettel mit Terminen für Kunden
Team	3 Team bestehen aus 3-7 Personen (Software, Firmware, Chemiker, Hardware); mehrere Produkt Owner und ein Scrum Master;	1 Team mit ca. 25 Personen (Firmware Entwickler, System Tester, QDAR, Design, Elektronik, Type Approval)
PIP	PIP mit Release plan/Scrum mit PIP sind komplementär; PIP wird als Regelwerk und als Paralleler Prozess verwendet	
parallel Entwickeln	Funktioniert; man kann Hardware abstractions layer parallel zur Schaltung entwickeln; Applikationen können während dessen entwickelt werden, wenn ein Evaluationsboard existiert	
Herunterbrechen von Stories	Funktioniert ganz gut, Konzept, Review von Bauteilen als Task	Hängt vom Team ab, dürfen selber entscheiden; es gibt Features die mehr als 1 Sprint in Anspruch nehmen und Stories welches 1 Sprint brauchen

Best Practices Interviews

Inkrement	ALADIN's Bezeichnung für ein Inkrement wird verwendet	wird nicht so dringlich gesehen; eher Ablauf besser zu gestalten; Vernünftige Ergebnisse sollen präsentiert werden nicht unbedingt fertige Versionen oder ein gesamtes neues Produkt
Bestellzeiten		möglich früh draufkommen welche Bauteile gebraucht werden (Schlüsselbauteile ca. 14-20 Wochen)
testen		Hardware on the loop zum Testen von Firmware und Regelsystemen (Typhoon Hil); verhalten des Gesamtsystems oder einzelne Teile darstellen; Simulationen bis zu einem gewissen Grad möglich