An empirical investigation of the requirements process at a successful company

Linus Ahlberg
linus.ahlberg91@gmail.com

Johannes Persson
ain09jpe@student.lu.se

Supervisor: Showayb Zahda, showayb.zahda@axis.com
Examiner: Krzysztof Wnuk (LTH), krzysztof.wnuk@cs.lth.se

2014-06-06

Master's thesis work carried out at Axis Communications AB
for the Department of Computer Science, Lund University.

Abstract. Requirements engineering is often seen as an important activity in software projects, since it impacts the projects' overall successfulness. However, with the introduction of agile methodologies, requirements engineering has gone through several changes. For example, rather than focusing on rigorous requirements specifications, agile methodologies advocate a more minimalistic approach where documentation, to an extent, is replaced by face-to-face communication. This trend is found at the company examined in this study, since the company utilizes a requirements process where the amount of requirements documentation is limited.

The aim of this thesis is to investigate the requirements engineering process used at the case company, Axis Communications AB. The investigation includes a description of the company's requirements process and the factors that facilitate the company's use of the process. Moreover, the benefits and the challenges of the process are explored, its scalability is examined and the viability of implementing a requirements database in the company's context is considered.

The methodology used in this study is influenced by Grounded Theory and data was collected mainly through interviews. Interview questions were created based on a literature review of current research in the field. In total, 16 interviews were held. Data obtained from the interviews was complemented with data from the introductory meetings and team meetings in order to form the base for the results and analysis presented in this thesis.

The study shows that the company's requirements process does not demand requirements documentation to be maintained indefinitely. Instead, requirements knowledge is shared in other ways, e.g. through direct communication between colleagues. This approach is, for example, facilitated by the company's culture and organization and yields many benefits and challenges. Currently, scalability does not seem to be a major challenge. Finally, this report concludes that implementing a requirements database is not a trivial task, claiming smaller modifications to the documentation approach could be a possible alternative for the company.

Keywords: agile, requirements engineering, lightweight, minimal documentation, platform development.

Acknowledgements

We would like to extend our warm thanks to examiner Krzysztof Wnuk for constructive feedback and support throughout the study, as well as Showayb Zahda for giving us the possibility of conducting the study in the first place and giving us support that helped conducting the study successfully. Additionally, our thanks go out to the interviewees of the study and the company in general, for a warm welcome and for always putting time aside for us when we asked for it. Without this support this thesis would not have been possible. Finally, we would like to thank Carmen Balsalobre for patience and support throughout the study.

Table of Contents

1 Introduction

The processes and activities within Requirements Engineering (RE) are often seen as critical for the successfulness of a software project. For example, studies have shown that inadequate RE is the main reason for software projects' failures [3] [34]. Consequently, much has been written on how to do RE in a good way. Although much work has been done, there is still controversy on how RE should be conducted in order to reap the benefits at a minimal cost. The undertaking of finding one process or a single set of practices that can be used more generally is, however, made substantially more difficult when taking the various contexts and needs of different companies into account.

With the introduction of agile methodologies, RE has gone through somewhat of a transformation. Instead of the rigorous requirements specifications advocated in the traditional, document driven software development, agile methodologies advocate a minimalistic documentation approach. Where traditional software development focuses on documentation and rigorous processes, agile methodologies focus on face-to-face communication and people. This has caused a change of direction in RE, not least within companies applying agile principles. Requirements specifications have, at least to some extent, been replaced with backlogs and burndown charts. Iterative development and frequent releases, applied in agile, have shortened the feedback loops, allowing customers to change their mind during the development process. Continuous prioritization of development activities has been utilized in an attempt to maximize business value. All of these changes have impacted the view of RE today.

As the area of RE is evolving rapidly, empirical research on RE processes in top-of-the-line companies are a useful source of new ideas and concepts. In this regard, the research in this study seeks to investigate the RE process at Axis Communications AB. Specifically, the process at the company is interesting from an RE perspective due to its limited amount of requirements documentation. Currently at the company, no pure requirements documentation is maintained after the end of development projects working towards a software platform. This approach creates several interesting challenges, which the company deals with through various methods.

This thesis explores the RE process at Axis Communications AB, treating the concepts and methods relating to this process. For the company itself, the study aims to give an overview of the process as well as an analysis of the benefits and challenges associated with it. For the research community, the study aims to present interesting concepts relating to the company’s requirements process and some explanation of the measures that the company is taking in order to facilitate efficient software development. Thus, the research questions for this thesis are the following:

RQ1: What constitutes the requirements process used at Axis Communications AB?

RQ2: Are there any factors at Axis Communications AB that facilitate software development with the current requirements process?

RQ3: What benefits does Axis Communications AB gain from using the current requirements process?

RQ4: What challenges does Axis Communications AB face due to the use of the current requirements process?

RQ5: What can be said about the scalability of the requirements process used at Axis Communications AB?

RQ6: Would the implementation of a requirements database be a viable option for Axis Communications AB?

The study was structured so that the scope would enable as much of a holistic perspective as possible of the requirements process within the company. However, the scope of this thesis is limited to the platform organization of Axis Communications AB, as well as, to some extent, other departments closely related to it. Therefore, the requirements process analyzed is the process as conducted by the platform organization at the company. Also, the process was studied as-is and information about any previous state of the process is therefore not treated in detail. The study examines relations between primarily development, product management and Quality Assurance (QA), but other activities have also been briefly examined.

Since this study explores the requirements process used at the company, documents which were considered not to relate to requirements are either not treated at all or only treated with little detail. Also, due to the aim of having a holistic perspective, no limitations have been put on what areas of RE that will be considered. In other words, any RE topic that is relevant to the company's requirements process has been taken into account. This includes the relevance of requirements for QA activities and how those activities are affected by the requirements process.

The structure of the sections in this report is as follows. Section 2 presents the results from the literature review performed in the study. Section 3 introduces the company and the requirements context. Section 4 treats the research methodology that was used in the study, giving some detail on the reasoning in choosing the specific method. Section 0 holds the results from the data collection and analysis phases. Section 5.4 summarizes the answers of the research questions and includes a discussion of limitations and future work. Finally, section 7 contains the conclusions of this study.

As this is a collaborative master's thesis, responsibility of the different activities was divided between the authors of this study. This division can be found in Appendix A.

2 Background and related works

This section aims to give a background and present the related works for the three main topics that were found to be relevant for this study. Firstly, the practice called agile software development is presented, including its main benefits and challenges. Secondly, the use of RE in software development is discussed. This includes exploring lightweight requirements as well as investigating both agile and traditional RE. Lastly, previous research within software product line engineering is treated.

2.1 Agile software development

The practice of agile software development, below referred to only as 'agile', is an alternative approach to traditional document-centric development [60], providing fundamentally different ideas about how to successfully develop software [9]. The reasons for the evolution of agile were many, including general resistance against document heavy processes and the challenge of changing requirements [14] [38]. When the speed of changes increased, caused by volatility factors such as customers not knowing exactly what they wanted, traditional development processes faced a major challenge [12].

Since the introduction, agile has grown popular in the software industry. This popularity is sometimes seen as a proof of the relative benefits of agile, but the methods within agile have also received criticism [38]. The success has been explained by the skill of the selected people using agile [22], and several weaknesses have been identified. The weaknesses include having insufficient documentation [30] [49], neglecting the need for spending effort on architecture and its specification [20] as well as not handling quality requirements, also known as non-functional requirements , adequately [24] [56].

There are many methods included in agile software development, e.g. eXtreme Programming (XP), Scrum and Agile Modeling. The different methods can be used as a basis to tailor a specific process after the needs in the current organization and/or task [5]. This is possible since all agile methods are not on the same abstraction level. For example, XP relates more to the daily programming work of a developer [8] [41] [42], while Scrum focuses more on how to coordinate and organize a team [57]. Other methods within agile include Crystal [55], Test Driven Development (TDD) [41] and Lean software development [39]. From these core methods, much effort have been directed towards extending and improving the methods to, for example, become compatible with software product line concepts [19] and handle various elements of the software process in a better way (e.g. agile prioritization [45] and agile release planning [29]).

The principles of agile software development have been presented and explored many times [5] [9] [41] [47] [49] [55]. The complete picture of agile software development is out of the scope of this thesis, but some of the most relevant concepts will be elaborated on. The principles revolve around elements such as working in an iterative fashion [36], having close interaction with the customers [43] and providing business value as early as possible in a project [13]. There are challenges, however, in how to interact with the customer representative(s) [42], the need for skilled practitioners [43] and how to handle the minimal documentation by spreading knowledge throughout the team during elicitation [55] and validation [27] activities.

One of the core principles of agile is the notion of self-organizing teams. This is advocated in the agile manifesto [9], and is also mentioned in both XP ("empowered teams") and Scrum [5]. The idea is to put trust into the competence of agile teams, thereby enabling them to tailor the process they follow after their specific needs and environment [32]. Researchers have claimed that self-organizing teams is an important factor for the success of agile projects [31].

Agile has been shown to have several benefits, although the view of agile as the best way of developing software has been debated. The advocators claim that agile methods increase the visibility of a project's progression past the early stages [13] and give the customer business value faster because of the iterative way of working [13], which has also been shown to reduce risks [41]. Other benefits of agile include the removal of unnecessary documentation by face-to-face communication [14], higher quality of the software, and increased customer satisfaction [41] e.g. due to the way agile embraces and adapts to changes in requirements [13]. The communication between business people and engineers has also been shown to improve when using agile practices [10]. Finally, a small increase of the methodology used in a project can impact the costs significantly [15]. Therefore, if people can communicate easier, e.g. through working in smaller teams, the development costs will decrease [15].

A lot of challenges and weaknesses have also been shown. The critics have pointed to the minimalism of documentation in agile, for example arguing that documentation reduces the knowledge loss when team members become unavailable [49] and that a lack of documentation is one of the main causes for fast deterioration of quality in software [30]. Additionally, motivated and skilled people, which are preferred in agile [9] [52], are not always available and can thus be used as an argument for more traditional processes [17] [52].

Furthermore, the agile methodologies have been shown to be difficult to scale, due to distributed development [49] and other scalability factors [5]. Some outright claim that agile methods do not scale [17] and that they only are viable in small teams [55]. The research that covers large-scale agile suggests that one way to handle scalability is to combine and tailor different agile methods with more traditional processes on a situational basis [5] and preserve company specific key processes [52]. The research

also deals with specifically scaling agile methodologies, such as how to coordinate several Scrum teams in parallel by dividing tasks from a master backlog to smaller team backlogs [52]. Ultimately, the large-scale implementation of agile software development is, however, still a major challenge in the industry. For that reason, case studies on successful large-scale companies exploring new possibilities in structuring agile processes are desirable.

2.2 Requirements engineering

This section treats lightweight requirements and the various approaches of working with requirements, specifically agile RE and traditional RE. As agile RE has a higher relevance for the work in this study, this topic contains a more detailed discussion of the practices and their implications as well as challenges within agile RE.

2.2.1 Lightweight requirements

During the literature review only one study [60] was found that provides a definition of "lightweight requirements". This study presents a number of criteria that defines "lightweight requirements". These criteria include that requirements are elaborated just-in-time, that techniques such as prototypes are used for eliciting and validating requirements and that requirements are iteratively validated in a certain manner. However, this definition is only applicable within the context of requirements documentation and not in the context of requirements engineering. Moreover, one study [26] describes "lightweight documentation" as an approach where documents are rarely maintained. This description is aimed at all software documentation, requirements included. Regarding the use of the term "lightweight requirements", another study [18] uses it without a definition. Instead the study presents a concept called "Story-Wall", which it claims to be a lightweight way of managing requirements.

Several other studies [23] [24] [25] [42] [56] uses the term "lightweight" in a requirements context. One of these [23] claims that requirements representation methods in non-traditional RE, e.g. user stories, can be considered to be lightweight. Also, several studies present different methodologies related to requirements which they claim to be lightweight [24] [25] [56]. One of the studies [56] proposes a methodology for eliciting and analyzing quality aspects. The authors of the study argue that the methodology is lightweight since it limits the number of quality aspects that are focused on rather than to document a comprehensive and detailed specification of quality requirements. Thus, it minimizes the amount of information that needs to be gathered about quality aspects.

The literature review showed that the term "lightweight" is not widely used in the RE community [23] [24] [25] [42] [56] [60]. Moreover, the occurrence of the specific term "lightweight requirements" is even rarer [18] [60] and only one of the studies [60] provides a definition of it. Instead, it seems like other terms are used in order to convey the meaning of lightweight. For example, one study [42] claims agile is a

lightweight methodology. Therefore, agile RE is interesting also from a lightweight requirements perspective. This is the reason for further exploring agile RE in the following sections.

2.2.2 Agile Requirements Engineering and its practices

Agile RE has in recent years spun out as an alternative way of doing RE [17]. The method evolved due to the inherent difficulty in specifying all the requirements upfront [60], which partly can be explained by the customer's inability to state its needs at an early stage of a project [12]. This leads to changing requirements in later stages of the development cycle [54], putting the project at higher risk [55]. Rapid changes are common in software projects [12] and are one of the most common reasons for project failure [55]. Agile RE mitigates this risk by working in an iterative fashion [60], which means that requirements emerge iteratively in short stages during development [44] [45]. According to one study [58], this means that the requirements specification should be released frequently in smaller independent and functional parts. Frequent releases also makes it possible to acquire customer feedback earlier in the project than when using traditional approaches and improve the customer's understanding of the software [60]. Other benefits of agile RE include better customer relations [54] and reduced scope creep [10].

Agile RE advocates practices such as continuous prioritization and prototyping [14] [46]. Continuous prioritization means requirements are prioritized between the iterations [14] [45] [46] [55]. The requirements are solely [45] [46], or at least mainly [14], prioritized according to business value, which means the features with the highest business value are implemented first. Prototyping is on the other hand used in order to elicit and validate requirements [60]. Through prototypes the communication with the customer is simplified and the requirements can be refined [46]. Also, the frequent communication between customer and development team, advocated in agile RE, acts as means of validating the product [46].

It is a fact that real life projects cannot provide complete requirements documentation due to limited resources [43]. Instead of aiming for perfect requirements agile principles focus on specifying "good enough" requirements [2], i.e. the realizable requirements which bring the most value to the customer [2] [58]. Furthermore, agile RE directs efforts to the requirements which yield the highest business risk [58].

In general agile RE handles this incompleteness by being less focused on documentation than traditional RE [22] [43] [55] [60]. Agile principles advocate face-to-face communication [24] and customer involvement in order to share knowledge [18] [17]. Research [15] has shown that face-to-face communication is more effective and efficient than documentation when communicating in small teams, thus reducing the overall development cost. However, the study does not identify the implications of having a focus on face-to-face communication in a larger scale. Another study [26]

shares the view of documentation as a communication medium, for example stating that there may still be value in outdated documents. The authors of the study argue that this is a reason to focus on that documents are easy to create rather than easy to maintain. The study proposes lightweight documentation, such as photos of white-board drawings, as effective means of documentation. Other lightweight documentation techniques include user stories, which have the objective of reducing the cost associated with requirements elicitation and requirements management [20].

There are different opinions regarding the amount of documentation needed in agile contexts. Some claim "well-written source code is self-documented" [55] and therefore no more or a minimal amount of documentation is needed [49] [55], while others argue that documentation of some sort is used in many of the agile principles [43]. Nevertheless, it is apparent that agile methods aim to reduce the amount of unnecessary work, requirements documents included [41].

One of the agile requirements documentation techniques is the product backlog. The backlog is central in some agile RE practices, since it works as a continuously changing requirements specification [43]. It is filled with high level requirements, such as user stories [43], which are used for communicating with the customer [46] [60]. In essence, user stories are scenarios written in plain text [20] that are placed in the backlog [45] and prioritized [17] by the product owner. Through inter-team communication the user stories can be broken down into more detailed requirements [17] [27] [60]. This involves the whole development team, which means knowledge can be shared without the use of documentation [27] [55].

One of the main reasons to why agile RE uses lightweight techniques such as user stories, is the wish of elaborating the requirements just-in-time, right before the actual implementation [10] [23] [60]. By this point, the understanding of the customer needs is better [60] and the requirements are less likely to change [10]. Not only does this reduce the amount of requirements documentation which becomes obsolete in environments of rapid change [17]. It also reduces the amount of time wasted in development trying to implement incorrect requirements, which can lead to unnecessary source code, higher complexity and higher costs associated with maintenance [55].

Agile RE has proved to be an interesting research field and many studies have been carried out in order to improve its applicability, effectiveness and efficiency. Methods improving the agile RE process in general have been proposed [47] [57], as well as several techniques supporting lightweight requirements documentation [17] [18] [30] [33] [49] [51] [60]. Other studies argue that the amount of documentation needs to be tailored according to the specific project and its context [15], and some even argue that certain agile RE methods only are applicable in certain contexts [43] [52] [55]. More documentation, with greater detail, is in general needed for new systems, newly formed teams, large or distributed teams and for systems with high criticality [13] [55]. However, based on the literature review performed in this study, not much re-

search has been done on how lightweight documentation actually is implemented in large-scale agile RE, and how the implementation of lightweight documentation affects the company. Therefore, case-based research on lightweight documentation in an industrial setting could bring value to the research field of agile RE.

2.2.3 The challenges of Agile Requirements Engineering

Many concerns have been raised about the agile RE way of working, both on a general level and relating to specific agile practices and methods. For example, the heavy dependence on customer interaction has been seen as a challenge [14] and some argue that agile methods assume too much about the level of contribution from the customer [43]. This relates to the fact that there is an inherent difficulty in going from high level user stories to implementation of actual code [52]. Also, while some solutions for showing the successfulness of agile teams have been proposed [30], managerial issues are still present because of the difficulty in getting time/cost estimates for larger pieces of software in agile RE [14] [32]. More specific issues within agile RE include that using prototypes can cause unrealistic customer expectations, as well as the challenge of using refactoring to effectively evolve an architecture over time [14]. It has also been shown that it can be difficult to get the development team involved in the development and management of requirements, together with a difficulty of getting the team to document the requirements [10].

Critics also argue that agile RE has a clear lack of documentation [30] [38] [42] [43] [46] [49]. This is sometimes believed to cause knowledge loss when team members become unavailable [43] [46] [49], but other [8] work claim that the knowledge loss is mitigated through pair programming due to the fact that multiple people gain knowledge of all source code. It has also been argued that a lack of documentation makes maintenance more difficult [30] and in many cases the maintainer has to turn to the source code or test cases in order to gain knowledge about the software [42]. About 40-60 % of the overall maintenance effort is believed to be linked to the task of understanding the software [51]. Also, the few documents that do exist may not be useful if they have not been updated along with source code during previous changes [51]. Since documentation can support maintenance, it is important that developers document code which is likely to cause future problems [41]. It has also been shown that different documents are needed during development and maintenance [51]. When maintaining software, documentation is used for giving an overview of the system at hand [51]. If the documentation is too detailed people will not update it, which generates out of date documentation [51].

Another issue associated to the lack of documentation is that it makes introduction of new team members more cumbersome [43]. New members will have many questions that good documentation could have answered [43]. Instead they have to ask

other team members, which slows down work for the development team as a whole [43]. This problem might be solved by using the test cases [27]. The test cases represent the system and can be used as some kind of requirements specification for new employees in order to understand the software [27]. Furthermore, if test cases are developed early, they can be used to validate the requirements and find new requirements which have been overlooked in earlier development [27].

Aside from the challenges relating to a lack of documentation, Agile RE has been accused for overlooking quality requirements [25], especially in early stages of development [17] [24] [38] [45]. For example, it has been reported that using prototypes may cause problems within areas such as scalability, security and robustness [14]. Inadequate attention paid to quality requirements may lead to several problems in later development phases such as bad software quality [13] or late architectural changes resulting in cost overruns and delayed projects [45]. The problem has been addressed by many researchers. Some say the problem solves itself, since the development team gets customer feedback after each iteration, including issues related to quality requirements [55]. Others stress the importance of using techniques for eliciting quality requirements early in development, before implementation [13]. Over the years, several methods have been proposed with the purpose of handling the problem of neglected quality requirements in a lightweight way [21] [24] [25] [56].

Thus, much has been said about the challenges which are linked to agile RE. These include maintenance issues, problems with introducing new employees and overlooking of quality requirements. However, more research is needed in order to see how agile RE is done in practice, e.g. about requirements prioritization [6]. Based on this, research might also be needed on how the challenges presented in this section are perceived by companies in the software industry. Therefore, case studies focusing on these aspects could extend the knowledge within agile RE challenges and countermeasures.

2.2.4 Traditional Requirements Engineering

Traditionally, RE has been broken down into four different activities, specifically elicitation, analysis, specification and validation [1] [43] [46]. Moreover, there is also the element of Requirements Management [59], which for example is concerned with the maintenance of requirements. In traditional software development, sometimes known as phased development [36], RE as a whole has ideally been conducted separately in the beginning of a new project with the aim of finding the complete set of requirements to be implemented [17]. This approach rests on the assumptions that mistakes found early are much less costly to fix, and that it is possible to find a complete, stable set of requirements in the beginning of a project [43]. In section 2.1 above, work has been presented that indicates that the second assumption does not

always hold true, causing traditional RE to suffer, e.g. in cases of market volatility [12] [19].

RE has traditionally been seen as a critical activity of software development [55]. This has resulted in a number of standards coming from the traditional RE viewpoint [55], as well as a large amount of literature on techniques and guides on how to write a good requirements specification [37] [59]. Some literature has listed a number of "good qualities" for a requirements specification [37] [43]. For example, a good requirements specification should, according to Lauesen [37], be "correct, complete, unambiguous, consistent, prioritized, modifiable, verifiable and traceable". However, the necessity of these qualities has been questioned. For example, it has been claimed that practitioners realize having a complete exhaustive requirements specification is unrealistic and that tacit (unspecified) requirements are necessary [37].

The requirements specification acts as a comprehensive description of what the specified software should do, containing both functional requirements and quality requirements [7] written on many abstraction levels [28]. However, the requirements specification is not the only used documentation that relates to requirements. Other documentation include architectural and design specifications, as well as testing documentation [26]. These are all interesting from an RE perspective, since they supplement the written requirements specification.

When comparing traditional RE with agile RE, a number of issues in agile RE are handled adequately through the traditional approaches. For example, the challenges of dependence on customer representatives, risks with minimal documentation or neglect for quality requirements are less significant in a traditional RE context [14]. Additionally, traditional RE gives greater guidance for developers on what to do, somewhat reducing the need for skilled people, which is an important aspect of agile [43]. One of the factors facilitating this guidance, namely documentation, might be more efficient than face-to-face communication in some cases. More specifically, documentation is efficient when the documentation replaces the need of having to explain the same thing to different people [43]. Regardless, agile RE has undoubtedly grown popular over the past decade [38]. Several studies have shown the general benefits and challenges of agile RE [7] [10], explaining the reasons for moving from traditional RE [7]. Additionally, some studies claim that the choice of whether to use agile or traditional RE depends on the situation [46].

While some statements has been made regarding the use of traditional RE practices in agile development [7] [14] [46], it seems not much research has been done on their practical applicability. Even though the core of traditional RE, with an emphasis on up-front specification, clearly does not synergize with the agile practices, smaller parts of traditional RE could possibly still be applied together with agile development successfully in the industry. The reasoning that different processes have to be used [46], or tailored [5], according to the current specific needs of a project/organization

is important in this regard. Basically, it means that the context in which RE is applied influences the choice of whether to include elements of traditional RE into agile RE or not. As little research has been found on this topic [5] [46], research on the practical implementation potential of RE practices in agile contexts is motivated.

2.3 Software Product Line Engineering

Software Product Line Engineering (SPLE) is a concept which is built on reuse of existing software when creating similar products [19]. The core reasoning of SPLE is that it can be economically rewarding to manage commonality in products that share similar features [40].For this purpose, SPLE advocates putting effort into "upfront long-term design" of the product line architecture [19] [40]. This provides a base for exploiting the commonality of the product line (also called the product family) by facilitating the reuse of common core-assets (the "platform" [40]) [19]. Managing and developing the core-assets and product line architecture is referred to as Domain Engineering (DE) [20]. To handle product variability within product lines the core-assets are extended and/or combined in different ways, thereby tailoring the functionality to address the needs of specific products [19]. This type of work is instead referred to as Application Engineering (AE) [20].

The main benefits of SPLE are that once a core-asset base is in place, the reusability factor reduces the development effort needed to create new (similar) products [11] [40]. This has shown to give multiple benefits [19], such as cost reductions and faster cycle times when developing new products. Among these benefits, flexibility to change is also presented. However, these changes need to be proactively anticipated when designing the core-assets base [19] [20]. Consequently, flexibility in SPLE refers mainly to "planned changes", that can be foreseen with enough certainty to be significant at the point in time where the core-assets are developed [19]. This means that the planning of the product line architecture and core-assets is associated with risks, since it needs a large initial investment in order to later provide the benefits of SPLE [20]. For these reasons, organizations using SPLE in volatile markets face a big challenge [19], and should thus adapt their processes depending on the market situation [40].

As a variant of SPLE, Agile Product Line Engineering (APLE) has gained more interest during the last years. This concept, sometimes referred to as Agile Software Product Line, focuses on combining elements between Software Product Line Engineering (SPLE) and agile software development [19]. The idea of the concept is to integrate principles from both fields in order to cover weaknesses in each of the approaches applied individually [20]. For example, agile software development faces a challenge in the scalability of its principles, while SPLE has difficulties in trying to handle volatile market conditions [19].

Studies have shown the combination of agile software development and SPLE principles, into what is known as APLE, to be both feasible [4] [11] [20] [40] and desirable [4] [19]. The general idea is that agile and SPLE complements each other. From the SPLE point of view, one of the core benefits of agile is that it handles changing requirements easily [19] [40]. Thus, agile can provide help when the market is volatile or when developers lack knowledge within the DE phase of SPLE, which both contribute to late and unanticipated changes [19]. On the other hand, agile has a big challenge in the scalability [5] [49] of the principles and methods, which is an area that SPLE might be able to provide support with [19].

The reasons and benefits for doing APLE are elaborated on by Diez et. al [19]. According to their work, the general advantages are:

- Reduced need of up-front investment in DE.
- Flexibility in volatile markets, where it is risky to commit to development of a large core-assets base that may (at least partly) become obsolete due to changes on the market.
- Ability to facilitate knowledge gain in cases where developers' knowledge of DE is lacking.
- Ability to use a combination of SPLE and agile software development (APLE) instead of applying them individually, meaning a larger variety of projects can use the method.
- Through use of agile, the possibility of reducing the time of the feedback between "RE, development, and field trial in innovative businesses".

Although the above studies have shown apparent benefits of APLE, there are challenges both on the higher levels (e.g. contradicting values) [19], as well as closer to the actual implementation of APLE. One of the challenges is the idea of lightweight documentation in agile, corresponding to the more document intensive approach in SPLE where documentation helps handling maintenance and evolution of the core-assets [4]. Secondly, there is an issue in that agile methodologies advocate a reactive approach, while SPLE instead leans towards a proactive approach because of the need of anticipating changes [4]. Thirdly, there are more implementation oriented issues in e.g. how to handle traceability and maintenance [19].

More specifically, how to handle the architecture is an important issue in APLE, as SPLE does much of it up-front while agile in general lets the architecture evolve iteratively [4]. The examined papers seem to treat this as one of the key challenges in this area [4] [11] [19] [20]. While the research question still is largely unsolved [20], the general idea seems to be to balance the both approaches, i.e. developing core-assets through a more lightweight up-front architecture [4], which then can be iteratively

developed as needed [19] [20]. The architecture can also be used as a frame, within which agile teams can solve issues in the way they prefer [11].

Because of these challenges, which relate to the successful implementation of the APLE approach, more research on APLE and how to successfully combine SPLE concepts with agile software development could be useful. This includes research on more specific topics such as RE within this context.

3 Case company

Axis Communications AB develops and manufactures network cameras, consisting of both hardware and embedded software. In recent years, the business has been growing fast and the company now operates on a global market as a market leader. The organization is made up from over 1800 people worldwide, from which a few hundred are involved in embedded software development. This software is developed towards an open platform, which causes the company to prioritize clean and stable APIs for its users.

The embedded software development follows an SPLE approach, which facilitates reuse of code in a variety of different products. This is done through the Linux Firmware Platform (LFP), which the code is integrated into. The platform itself contains all functionality (of the products it is intended for) and uses configuration settings in order to turn certain functionality on and off. Thus, the platform can satisfy the functionality needs of many different products, using different configuration settings for each product. In essence, each product has its own functionality configuration as well as its own hardware capabilities. This creates a certain complexity in the development of the LFP.

At Axis Communications AB, development of the platform and development of specific software functionality for new products are divided into two different organizations, as can be seen in Figure 1. The New Video Products department (NVP) uses the functionality available on the platform when developing products, striving to minimize the amount of software changes done before integrating the software back to the platform. However, some additional development is in many cases necessary in order to provide new products with new functionality. In this case, development is done in the product development department and integrated into the platform after the product has been released. Thus, new functionality flows into the platform not only from the department focusing on platform development, but also from NVP, creating interdependence between the departments.

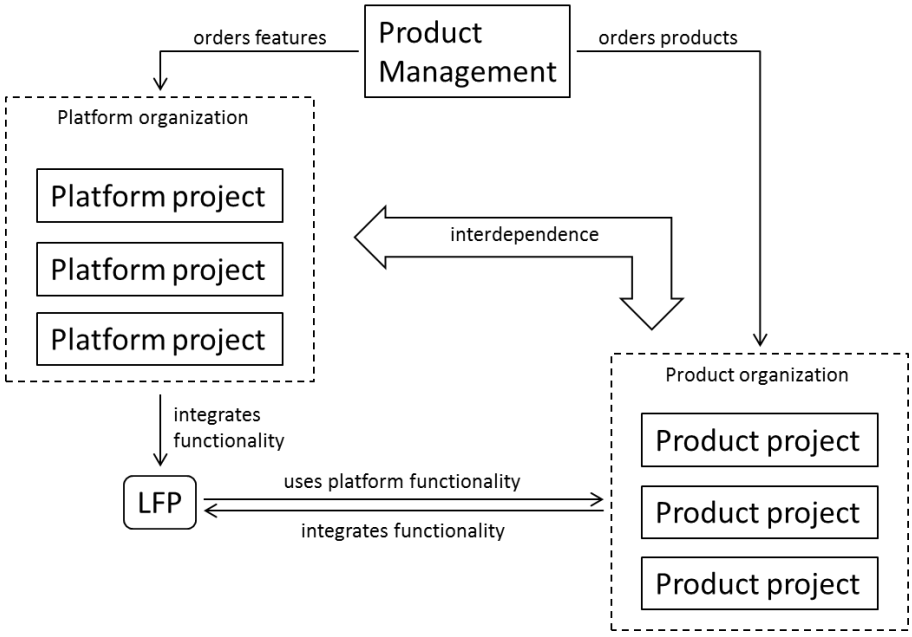

Figure 1. The flow of functionality to the platform.

However, the main focus in this thesis is the platform organization, looking into how its different parts perceive and are affected by the amount of requirements documentation available for the platform. This includes:

- Product Management – responsible for channeling market needs into requirements for new software.
- Development teams – responsible for developing and maintaining the software platform so that it meets the requirements.
- QA – responsible for testing the functionality on the software platform.

In total, the platform organization consists of 125-150 people that are co-located at the company's main site. From these, the majority works in development teams, generally consisting of up to 10 people. Different teams are responsible for different functional areas and each team mainly conducts projects in their own area. Because of this division, the teams are referred to as "function teams". The areas are for the most part architecturally separated in order to limit the number of teams that are affected by a project. One project manager and one technical lead are assigned to each project, carrying the responsibility of leading the development work. Moreover, each project has at least one assigned QA resource that participates in the project's activities and writes test cases for the output of the developers.

For the most part, agile methods such as Scrum are used by the teams within platform development. Agile methods are also used to a large extent by all departments conducting development for the company's market[1]. The projects in platform development follow predefined guidelines for the development process (including how requirements are handled), but are allowed to do changes to the process if the changes are approved by all project stakeholders. This, in combination with the fact that the process itself is continuously evolving, means different teams are working somewhat differently. Sometimes they even work with different types of documentation, altogether adding to the complexity of this study.

The responsibility of existing code is divided into two layers, namely Code Block Architects (CBAs) and Code Block Maintainers (CBMs). The CBAs are responsible for larger components, where they manage the overall architecture and design of their areas. Below each CBA there are several CBMs, where each CBM is responsible for a smaller portion of code, e.g. being responsible to review any changes made to these code blocks. The CBAs and the CBMs that are tied to one functional area are normally part of the function team that is responsible for that area, but exceptions exist.

In general, the culture at the company leans towards spreading knowledge about requirements verbally rather than through specific requirements documents. The company values an open climate and encourages its employees to be communicative. Informal communication is promoted, e.g. making it culturally accepted to ask questions and have discussions without booking formal meetings. Thus, much knowledge is shared through discussions, even between departments, and helpfulness and team spirit are emphasized.

3.1 The requirements context

This section describes the parts of the company's processes that are related to requirements capture, prioritization, documentation and validation. As there is no explicit "requirements process" for the platform development at the company, much of the content in this section is based on the more general software development process. The data in this section was extracted from company specific process documentation as well as meetings and interviews that were held during the study.

3.1.1 The requirements process – an introduction

[1] For more details see the study, available internally on Axis, by Jan Bosch & Helena H. Olsson (september 2013): "Development Practices at Axis Communications".

As can be seen in Figure 2, the business strategy and the roadmap work are the starting points in the requirements process. At the business strategy level, senior management decides what is important for the company to focus on, setting the frames for the later stages in the requirements process. In the next stage Product Management creates a roadmap, where ideas that have been brought up are prioritized against each other based on the business strategy. This process is repeated regularly in order to keep the roadmap up-to-date. The roadmap is the source from which orders are written. Each order represents a specific assignment formulated as high level requirements and forms the basis for the work that is conducted in a project. Orders are written for work on both features and architectural enhancements. The orders are then given to projects, where each project has one order and one orderer. The orderer's role is to answer questions and take decisions when it is needed in the project. The orderer is thus the main source of assignments (requirements) for the team and can be a person from Product Management or an architect (CBA or System Architect) in the development organization.

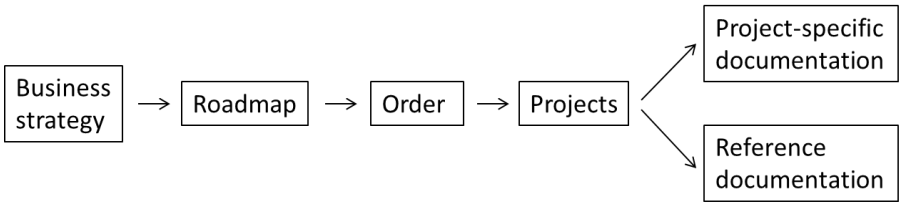

Figure 2. Overview of the requirements process from strategy to documentation.

During the course of a project, the requirements are broken down and implemented. In the project, resources from QA are participating in team activities such as daily standup meetings or workshops, writing system level test cases for the project and to some extent participating in breaking down the requirements. Continuously during the project, the team writes both project specific documentation and reference documentation. However, when the project has been closed the project specific documentation is not maintained anymore. Instead, the company focuses on maintaining test cases and reference documentation. Here, the reference documentation aims to specify the design and functionality for the LFP.

The existing specifications of requirements are largely contained in the different project specific documents, where several different documents (e.g. order, SWO and PRS) may contain requirements. As the project specific documentation is kept at different project sites and not maintained after the end of the project, the company does not have a maintained collection of requirements. The written requirements in the organization are not stored in a requirements database and the requirements documentation is not kept in a common repository.

In practice other communication channels play an important role at the company, giving employees some understanding of what constitutes the requirements even without a comprehensive requirements collection. This will be further elaborated on throughout the report.

3.1.2 The interface between orderer and team

Because the requirements in the order generally are on a high level, the team receives additional input to help them develop software that is in line with the orderer's wishes. Thus, the order is merely the starting point, after which the team needs to have additional contact with their orderer. In practice, the orderer often participates in planning and status meetings as well as other activities performed by the team, in order to be able to answer the team's questions and follow their progress. Additionally, the orderer reads some of the project documentation to ensure that the team's interpretations are correct.

The close contact between a project and its orderer is facilitated by an iterative and collaborative setting, where the team and the orderer both participate in activities such as breaking down requirements and planning. The work with planning and scoping is most intensive in the early phases of a project, where pre-studies, workshops, informal discussions and other activities can be performed. For example, this is done to work out what the project should achieve, what conceptual solution should be implemented and which assignments should be done first. From this, the team can estimate how much time it will need to implement what the orderer wants from the project.

Through iterative planning the tasks are prioritized and ordered with the purpose of maximizing business value. In other words, the most important tasks of a project are performed first if possible. The iterative process also pressures the teams to deliver functioning software that the orderer can examine and verify, ensuring that it reflects the underlying market needs. Typically, this is performed through having a demo at the end of each development cycle.

Even after the work in the planning phases of a project, the scope of a project is always subject to change. It is up to the orderer to decide if a project's scope should be extended or if it should be narrowed as the project progresses. Big changes to the project's scope or changes that will delay the delivery of the project significantly are, however, escalated to a steering group who then makes the decision for the project. In a similar way, project time frames are handled, depending on how the orderer feels about the project's performance. For example, a delayed project that is considered to be doing valuable work will generally be allowed to carry on. Also, due to the iterative work, a project that is closed before completion will generally have done partial deliveries that are of value to the orderer, even if the initial scope was not completed.

3.1.3 More details regarding documentation written and used in projects

This section is mostly a walkthrough of the different documentation being used at the company. Certain understanding of the documentation is preferred for the later parts of this thesis. Table 1 gives an overview of the documents with the highest relevance for the study. The rest of this section discusses the documents in more detail. The order was described in the above section and will therefore not be treated in detail again.

Project specific documentation refers to documentation produced in the project that is not maintained after the project has ended. Reference documentation is on the other hand not project specific, even though the documents might have been originally created in a project. The idea is to keep the reference documentation up-to-date, reflecting the current state of the software.

Generally, the different documents in Table 1 are written in different formats, e.g. .doc, .pdf and.xml. The documents are stored in various places in the organization, i.e. there is not one place that holds all the different documents. Some documents are also copied and stored in several locations, for example both on project specific intranet pages as well as in the company's revision control system, Git.

Table 1. A summary of the most relevant documentation at the company.

Project specific documentation	
Order	The order is the initial written requirements that form the first input to a project. Generally the requirements are on a high level.
Software Overview (SWO)	The work with the SWO is conducted in order to get an overview of what the project needs to do to understand its assignment. For example, needed pre-studies are specified along with the architectural areas that are affected by the project.
Product Requirements Specification (PRS)	The PRS is a requirements specification for the scope of the project. The PRS documents can look differently, depending on where in the organization the document was created.
Backlog	The backlog is used as a substitute to the PRS, currently being adopted by a few teams at the company. The backlog can be owned by the orderer, who fills it with prioritized work items. The team can then for each development cycle implement as many of the highest priority items as possible.
Reference documentation	
Platform Functional Description (PFD)	The PFD is a functional description of the behavior of the software. While it specifies details about functionality, it is not a requirements specification in the traditional sense. Most of the teams have used the PFD at some point.
Capability Description (CD)	The CD is also a functionality description, on a similar (but not identical) level to the PFD. The main differences are that the document is owned by Product Management and that it is more market oriented than the PFD. Only some teams

	have used this document yet and even fewer have used it after recent changes.
Platform Architectural Description (PAD)	The PAD is an architectural description, giving developers some details about the design of different components in the software. This document contains no requirements, but it serves to provide an understanding of how the system works.

Here follows an elaboration on some of the different documents, starting with the project documentation. The SWO is an overview document of what the project should do during its lifetime, including some kinds of early requirements. It also contains information about the different activities that need to be performed, what architectural areas are affected and who the responsible CBAs and CBMs for those areas are. Also, the SWO specifies what the project does not know and what needs further investigation (e.g. through pre-studies). The document is in itself used as a tool for planning, since it gives an overview of the project. It can also be read by the orderer to make sure that the project's progress is satisfactory.

The PRS is a project specific specification of requirements, meaning it describes what a project should achieve. The responsibility for creating and maintaining the PRS during the project is taken by the project manager. Aside from pure requirements, the PRS can also specify the project's deliverables. This document is used in many parts of the organization and while there is a template for it, different parts of the organization and different project managers choose to write it somewhat differently. A few orderers/teams have adopted the use of a backlog, replacing the more formal and detailed PRS. The backlog can be owned and updated by the orderer, in contrast with the PRS which is owned by the team.

Regarding the reference documentation, the PFD is a description of the current functionality, e.g. including web GUI mockups where applicable. Its description of functionality is on a relatively high level and from a technical point of view. The PFD is associated with its functionality and should, according to the process description in the company, always be maintained. Corresponding to the PFD is the CD, which also is a functional description. The CD is one of the newer documents and has so far only been introduced in a relatively small scale. While the CD to some extent shares the characteristics of the PFD, they are also different in a number of aspects. For instance, the descriptions in the CD are to a larger extent from a market perspective. Also, the scope of the documents can differ, with regards to the amount of functionality described. Moreover, the PFD is maintained by the teams who change the functionality, whereas the CD is owned by Product Management.

Even though most of these documents are created and maintained in the development organization, other departments use them for their own purposes. For example, QA uses the PRS and the PFD when writing test cases for the functionality implemented in a project. Also, Technical Information Management uses the documentation when writing help pages, user manuals or other public information. The usage of

documentation in several different departments creates a certain complexity in understanding the purpose of the different documents, something that is discussed later in this report (primarily in section 5.10.3 “Understanding the big picture”).

4 Research methodology

The research in this thesis is exploratory, conducted as a flexibly designed case study with influence from Grounded Theory [16]. As the primary data collection method, semi-structured interviews were used. Interviews, generating qualitative data, give richer results compared to quantitative methods (such as surveys), which instead give more precise data [50]. Thus, the interview strategy was chosen, with semi-structured interviews as the main data collection method. Semi-structured interviews were chosen with regards to the exploratory nature of the research, giving some flexibility regarding the precise questions asked. This research design, including its general characteristics, is in line with the primary research strategies used for case studies [50].

This section consists of two main parts, where section 4.1 presents Grounded Theory as a research methodology and how it was modified in order to make it adequate for this research. Furthermore, section 4.2 presents and elaborates on the main activities that were conducted in this study, including pre-study, data collection, data analysis and validation.

4.1 Grounded Theory and modifications

The methodology of this study emanated from a Grounded Theory perspective, which is a research methodology developed “for the purpose of building theory from data” [16]. The method was chosen due to its flexible, yet rigorous, design. This section introduces Grounded Theory together with the modifications that were necessary in order to use it in this study.

Grounded Theory seeks to generate well founded (grounded) theories through collection and analysis of data in a certain fashion [48]. Two of the core points in Grounded Theory that were also applied in this study, are theoretical sampling and constant comparison [16]. Theoretical sampling means collecting data (samples) and analyzing it iteratively, allowing flexibility in the choice of data sources and data collection methods in order to aid the researcher in developing the theory [16] [48]. Constant comparison instead refers to the notion of always going back and forth in the data, asking questions about the data as they are thought of [16]. This notion facilitates the evolution of a deeper understanding obtained through theoretical sampling, where new questions can be answered continually through additional data collection.

This process repeats itself until the researcher feels that the theory is properly grounded in the data and that new interesting pieces of information cannot be found [16].

For this study, Grounded Theory was deemed appropriate because of its exploratory nature, where theoretical sampling could be used to gradually increase the level of understanding of the studied phenomenon. It also enabled a more flexible collection of data, since new questions could be asked in each interview. This approach significantly increased the depth of the data as a whole and enriched the researchers' understanding of both the context and the phenomenon. This also resulted in that the interviews became less and less structured as the work progressed, as this was found to give the richest data. At the same time, as more facts were uncovered, the initial focus started to shift.

For the analysis of data, Grounded Theory uses a coding scheme including three stages, namely open coding, axial coding and selective coding. Here, the purpose of the open coding is to find concepts, which have a certain set of properties, as well as dimensions along which these properties can vary. The next step is to perform axial coding, which is the activity of finding the relationships between the concepts. Finally, selective coding is the stage where a single core concept is decided upon, a concept which should be tied to all the other concepts and give the theory as a whole the biggest possible explanatory power. [16]

Due to the exploratory nature of this study, after some initial interviews it became clear that the wide scope of the obtained data prevented the generation of a manageable set of concepts using the pure open coding scheme in Grounded Theory. This called for modifications of the coding procedures. Additionally, individuals were found to perceive the process in different ways depending on where in the organization they worked. Without the modifications that were applied to Grounded Theory, this could have made the interpretation of data more complex.

The modification in this case was to generate statements instead of categories, concepts, properties and dimensions, as prescribed in Grounded Theory. This narrowed down both the amount of data in the codes and made it possible to compare viewpoints efficiently, but did not negatively impact the scope of the work. Having statements as the main unit of analysis also enabled comparison and contrasting of different viewpoints, while at the same time getting information on what the process actually looked like, as well as what variations existed. It also allowed for direct corrections of the obviously incorrect results, only recognizing that some interviewees had incorrect perceptions of the matter. In essence, this modification allowed the authors of this thesis to react to the data continuously and only use the parts relevant for the research at hand. Using these statements also worked as a way of condensing the data in each interview, giving an overview of the complete data set.

Because the coding scheme was modified with regards to the open coding, neither axial nor selective coding were applicable. However, full adherence to Grounded

Theory was not necessary since the goal of this study was not to derive cause-effect relationships or to build a generalizable theory. Nevertheless, inspiration from the concepts in Grounded Theory was used for further analyzing the data. For more details regarding the data analysis, read section 4.2.3 below.

4.2 The main activities in the study

The research was conducted in four activities, which are described on a greater detail below. Even through the activities are divided into separate sections here, they were not completely sequential in practice due to the procedures of Grounded Theory research. The following activities were present in this study:

1. Pre-study, which consisted of a literature review and an initial study of the company, including its requirements related processes and organizational structure. The initial design and reviews of the interview instrument were also performed here.
2. Data collection, which consisted of choosing interviewees, conducting interviews, transcription activities and other work relating to obtaining and extracting the data.
3. Data analysis, which consisted of generating statements from the data, comparing these statements between the interviews and finally extracting more profound findings from the data.
4. Validation, which consisted of various steps taken in order to validate the findings.

4.2.1 Pre-study

This phase consisted of three main sub-activities, namely a literature review, a study of the company as well as the creation of the interview instrument. The first two activities were performed in order to provide a knowledge base, both of research within the relevant topics and the company's organization and processes. This supported the creation of the interview instrument, where the extracted knowledge was condensed into a practical tool to help structure the interviews.

The literature review was carried out to explore previous research relating to the context of the company. Primarily, the sources used for finding this research consisted of the electronic databases Engineering Village[2] and IEEE Xplore[3]. Snowball sampling was then applied in two rounds, first on the base set of discovered literature and then on the additional sources found during the first round.

[2] http://www.engineeringvillage.com
[3] http://ieeexplore.ieee.org/Xplore/home.jsp

Several different queries were used when searching the databases. The queries were derived from the context the company was operating in and the topic of the case study. The result of each query as well as the work of exploring references of existing papers were documented so that it would be possible to evaluate the best search words and the most promising areas to do additional queries within. The keywords were chosen in order to find studies in the following research fields: large-scale agile development, RE, SPLE and lightweight documentation. The scientific papers were evaluated based on their general relevance to the case study and their publication date.

The papers found in the literature search were reviewed and the relevant content was extracted into a spreadsheet. The spreadsheet contained the following categories:

- Prioritization based on relevance for the case study
- Main findings
- Findings related to documentation
- Useful quotes
- Limitations
- Challenges

Approximately 3000 scientific papers were found during the query searches. Of these, over 650 papers were further examined. This yielded 23 relevant papers that were summarized according to the above categories. Also, 21 more papers were found and summarized during the snowball sampling. Additionally, 16 new references were added during complementary literature searches. The final set of references included 60 items that were used in this study, of which 20 were journal papers and 18 were conference papers.

Next, the company study was conducted in order to gain insight into the company's organization and processes. This was complemented with introductory meetings across the organization, which resulted in a deeper understanding of the different roles in the organization and how they relate to each other. In total, 17 introductory meetings of approximately 15 minutes were held. Also, several team meetings were attended in order to gain knowledge in the team's internal processes.

The purpose of creating the interview instrument was to structure the interviews and provide guidelines for their execution. The instrument was based on the research questions and formulated and structured depending on the interviewee's specific department and role. This was possible due to the knowledge gained during the company study.

The questions in the interview instrument were continuously refined and rephrased. Furthermore, their adequacy was reviewed by both the supervisor and the examiner. The reviews resulted in a restructuring of the questions in order to better manage the amount of questions and their scope.

After the initial reviews of the interview instrument, four pilot interviews were held with the purpose of testing and evaluating the questions. Based on the results from the pilot interviews, the interview instrument was reworked and restructured once again in cooperation with the examiner. The results of the pilot interviews were included in the data analysis. The final interview instrument, connected to the specific departments and roles, can be found in Appendix B.

4.2.2 Data collection

The interviewees were chosen from different departments and teams, with the purpose of getting a wide perspective of the company's processes. The choices of interviewees were based on the knowledge of the organization as well as recommendations and information obtained during introductory meetings and other interviews. Caution was taken to find interviewees with different roles and experience, in different places of the organization, in order to get the most comprehensive and reliable data possible. It was stressed at the beginning of each interview that the answers were going to be anonymized and not shared outside the interview. For use in this report, the responses were first anonymized through the analysis process.

In total 16 interviews were held with different roles, see Table 2. Note that in this table, the use of "senior" in the Role column only is based on whether or not the interviewee has worked more than five years in the current role at the company. On the other hand, the General experience column conveys the general amount of experience in that role, thus not only limited to experience at the company. Also, the organizational distribution of interviewees is depicted in Appendix C.

Table 2. Interviewees' roles and experience.

Code	Role	General experience (years)
Da	Developer	13
Db	Senior Developer	8
Dc	Developer	11
Dd	Developer	10
De	Senior developer	9
Df	Developer	4
Dg	Senior developer	9
Dh	Developer	3
Di	Senior developer	5

Dj	Developer	6
PjMa	Senior project manager	7
PdMa	Product manager	1
PdMb	Senior Product manager	6
Ta	Senior tester	9
Tb	Tester	8
Tc	Tester	4

Each interview was booked for one hour each, but lasted between 36 minutes and 73 minutes. The interview instrument was used only as support for what areas to focus on rather than as a strict checklist. Additional questions, i.e. questions not specified in the interview instrument, were asked in order to elaborate on interesting themes or to explore unanticipated areas during each interview. The discussions were quite open in order to keep the responses uninfluenced and, thus, more credible. As the understanding of the company increased, the interview instrument was not followed as strictly, in accordance with theoretical sampling in Grounded Theory. The interviews were recorded in audio format to enable a more comprehensive analysis of interviewee responses in later phases of the study. All interviews except one were held in Swedish.

After each interview, the answers were transcribed based on the recordings. The length of the transcriptions varied between 2193 and 6570 words. These summaries were stored together with additional notes about each interview and each interviewee, e.g. experience, department, role. The summary of each interview was sent by mail to the interviewee who was given an option to read it and give additional comments. The team meetings and the introductory meetings that were seen as relevant for this study were also transcribed and used.

4.2.3 Data analysis

Statements were used as the main unit of analysis, see section 4.1 “Grounded Theory and modifications”. The statements were extracted from the data based on relevance and credibility, where the formulation of the statement reflected its characteristics (e.g. whether it was an opinion, an experience or a suggestion, as well as the interviewee’s certainty and other properties) and the surrounding contextual information (e.g. earlier statements, tone of voice, position in the organization). These statements essentially acted as summaries of the data and could then in later stages be compared and contrasted against each other efficiently. Statements were extracted continuously as the data was obtained and all interviews as well as the relevant meetings that had been summarized in text were included in the final set of statements.

As the amount of data grew and larger patterns could be identified, the statements were sorted into different sets based on department and function (platform development organization, product management and QA). In these sets, the statements were then sorted and structured in different ways. Essentially, combinations of (lower level) statements made up new high level statements, bringing the lower level statements up to a more abstract level. To avoid ambiguity, these higher level statements are referred to as assertions. The assertions were, just as the statements, formulated in order to reflect the content of the underlying data (in this case the statements themselves), making each statement have an influence on the formulation of the assertion. At this stage, care was taken to assure that the assertions had a high degree of "correctness", e.g. giving assertions based on only a few statements a less distinct formulation.

Since the assertions were derived from the statements from one department at the time, the assertions could be used to compare the viewpoints of the different departments. This was useful in order to assess if employees from different parts of the organization generally had different perceptions of the process. In total, analyzing 861 unique statements generated 260 assertions. The assertions can be found in Appendix D, but for anonymity reasons the specific underlying statements will not be presented in this thesis.

During the data analysis, the assertions were both used as pieces of data and as an index, allowing the researchers to find the underlying data on a specific topic. The use of the condensed assertions gave the researchers a good overview of the data, aiding the task of structuring the data into an outline. From this outline, detailed sections were then written, where the authors could further investigate the assertions' underlying statements as well as the interview transcriptions. Thus, having full traceability between assertions, statements and interview transcripts, navigation through the data was significantly enhanced. This, in turn, allowed going back and forth in the data, making comparisons between the assertions, statements and the transcriptions. These comparisons also ensured that no erroneous interpretations were made – much in the same spirit as the concept of constant comparison in Grounded Theory, presented in section 4.1.

4.2.4 Validation

The results of the report were validated through several steps. Firstly, the supervisor continuously reviewed the report as it was written. Secondly, the results of the report were presented at the company, where a number of employees attended. In connection with this presentation, open discussion of the results was also held.

5 An elaboration on the company's requirements process

This section presents the data regarding the company's requirements process that was collected and analyzed during the study. In the light of the data, potential implications and any conclusions that can be drawn, relating to the research questions, are also discussed. The structure of this section is elaborated on in Table 3.

Table 3. A summary of the different sub-sections in this chapter.

Sub-section	Description
5.1 The order and the orderer	Discusses the current state of this interface as well as the implications. The section brings up the characteristics of the communication between the orderers and the developers as well as the nature of the order. The high level of the order is elaborated on and some reasons for having this level are presented.
5.2 The use of available sources for requirements knowledge	Reports on ambiguity regarding what is considered to be requirements at the company. Discusses the use of different sources that may be used in a communication of requirements knowledge. These sources are, for example, documentation, test cases and the products themselves, as well as sources like colleagues and the code itself.
5.3 Quality requirements	Presents some details about how the company handles quality requirements and what implications this approach has. The general theme is that quality requirements in many cases are not specified, causing QA to have some difficulty when writing and running their test cases.
5.4 Benchmarking as requirements	Describes a concept for specifying requirements that is used at the company. Essentially, the requirement relates to an existing artifact, for example saying "this version should not be worse than the old one". Challenges with the concept are also brought up and discussed.
5.5 Implications of the company's requirements process	Discusses four topics, namely: 5.5.1 Understanding the intended behavior of a feature, where the issue of determining how a piece of software "should" behave is discussed. Also treats the significance of incorrect tickets and their consequences. 5.5.2 Understanding which functionality is included in a piece of software. Difficulties in this regard were experienced in both QA and Product Management. The scope and consequences of the difficulties are discussed in this section. 5.5.3 Complementary information about requirements, where other information related to a requirement except the requirement itself is discussed. This information can, for example, be information on what importance the requirement has, who ordered it and what the rationale was for ordering it. This information can be important from several aspects, as described in the section. 5.5.4 Choosing what test cases to run. QA experiences some difficulty in choosing test cases for a piece of software. Underlying causes are brought up and explained, for example low traceability between tests and requirements/products and incomplete data from old test runs.
5.6 A remark on the general quality of the software	A short elaboration on how the quality of released software is perceived by the employees at the company.
5.7 The scalability of the requirements process	Discusses this topic, as well as what activities the company currently is performing to deal with scalability. To summarize, this study cannot report on any problems that are significant from a scalability point of view. Instead, the inter-

	viewees tended to associate scalability with lightweight processes.
5.8 Ongoing improvements	Describes the improvements currently in progress at the company that are relevant for this study. These improvements consist of work with automated tests and automatically generated feature lists. Also discusses the impact of these improvements.
5.9 Communication channels	Discusses the use of face-to-face communication and documentation as communication channels at the company. Specifically treats the benefits of using face-to-face communication and when it might not be appropriate, as well as the main purposes of documentation, from an RE perspective, that were found in the study.
5.10 Soft factors	Brings up other, less tangible topics, namely: 5.10.1 New employees, which gives an understanding of what the process of introducing new engineers looks like and whether or not requirements documentation is critical for this introduction. 5.10.2 Knowledge sharing, elaborating on how knowledge is shared between individuals. Also treats the challenge of being dependent on knowledgeable individuals. 5.10.3 Understanding the big picture, where the emphasis on the employees at the company to understand the "wide" perspective is brought up and discussed. The concept is explained further and the implication of having a clear focus on it is elaborated on in the section.

5.1 The order and the orderer

The orders are, as previously mentioned, the starting point for a project, containing the project's assignment in written form. Two developers and one project manager (Db, Dg, PjMa) found the order to be vague or unclear. As some developers (De, Dg, Dh, Di) put it, the orders are on a high level, thus containing relatively little detail of what should be done by the project. To make up for any uncertainties in the order, the teams discuss the order iteratively and in close contact with their respective orderer. One senior product manager (PdMb) and one developer (Dg) indicated that orders originating from CBAs and System Architects generally are written on a more detailed level. However, one developer (Dd) pointed to that unclear orders are not necessarily a significant problem. Additionally, a project manager (PjMa) claimed that breaking down the high level requirements through discussions and close contact with the orderer facilitates knowledge sharing. This corresponds to statements from both interviewed product managers (PdMa, PdMb), who confirmed that they tend to work closely with their projects in order to aid the projects' understanding and make sure their interpretations are correct.

One of the senior testers in QA (Ta) also claims that having high level orders actually can be beneficial from an organizational point of view:

> *"Nobody knows how it [the software] should work, really, and maybe that is fine. I don't think it is a good idea to script everything in detail. That people need to talk to each other and maybe be creative is perhaps better than to always be super clear."*

- Ta

The high level of the requirements in the order has, however, seemingly increased the dependence between the orderer and the project team. This is also indicated by a developer and a product manager (Dg, PdMa). Experience might be one of the factors influencing the dependence level, as implied by PdMa:

> *"The people I work with at [department] are great, they are senior enough to be able to perform the task themselves. When I finally have managed to explain what it is that I want to achieve, they are very independent. It is a bit worse over at, mainly, the [platform department]. They are more used to 'you should write this thing like this' and then I come with something completely different. In this case, it is very much up to the project manager to do the job and get the project to understand what they should do. […] I think I am more deeply involved in those departments, they need more input."*
> - PdMa

The product managers that were interviewed (PdMa, PdMb) indicated that this dependence is higher in the beginning of projects, when the team needs someone to explain to them what the high level requirements are. Currently the dependence between orderers and their teams is perhaps not an issue on a daily basis, due to the close contact that these parties have. However, the dependence on individual product managers might not be ideal from a long term perspective. In other words, a sudden loss of a product manager could mean significant impact on the work in the corresponding teams. Nevertheless, according to the company's annual report for 2013 the personnel turnover of the company was less than 6% for that year, which a representative from HR claimed to be low compared to other companies (also indicated by previous work [35]). As turnover affects the overall significance of employees leaving the company, a limited turnover might reduce the challenge in having dependence on product managers. Additionally, previous research indicates that a significant dependence on the "customer" is an inherent trait of agile RE [14] (see section 2.2.3 "The challenges of Agile Requirements Engineering"). Therefore this dependence on the product managers must be weighed against the benefits of using agile RE.

According to one of the product managers (PdMa), writing detailed technical requirements in the order is neither seen as feasible nor desirable, from the perspective of Product Management. Instead, the product manager prefers to write specifications from the problem domain (what the product should be able to do) rather than how the problem should be solved (how the software should be changed to satisfy the needs). The fact that orders are generally written on a high level can partly be explained by

difficulties for product managers to know exactly what is needed from a technical solution at an early point in time. Hence, one product manager (PdMa) favors flexible solutions, such as agile development, where it is possible for the orderer to change the requirements as the project progresses. This flexibility is also facilitated by having a high level order, since a high level order makes it less troublesome to change the requirements during the project. However, the flexibility of changing requirements during the project also might result in deterioration of the accuracy of any estimates done earlier in the project. Depending on the criticality of the estimations, this could mean unwanted consequences. For example, several interviewees (Dg, Di, PjMa) pointed to that incorrect estimates have caused difficulties in achieving the planned integration dates, which is an issue since the company uses fixed integration slots for the different teams. Thus, a delay for one team could possibly make other teams also run late in the integration phase, as noted by one interviewee (PjMa).

A senior tester in QA (Ta) as well as one senior product manager (PdMb) expressed worries that individuals and teams may interpret details in a specification too literally, either through taking something to the extreme just because it was written in a specification or through over implementing the details just to get the system to behave exactly like the specification says. The product manager (PdMb) reasoned that through letting a development team, being specialists in their specific area, work out the solution that they find to be most appropriate, time and effort of implementation may be reduced. In order to do this, the product manager thought it is helpful if the developers understand the bigger picture of what they are developing, including the perspective of both customers and other users (more on this in section 5.10.3 “Understanding the big picture”).

Many interviewees (Da, Db, Dd, De, Dj, Dg, PjMa) reported that they use demos in their projects, a concept that helps the orderer to give early feedback on technical solutions. In other words, demos give the orderer an idea of how the work in the development team is progressing and how the current state of the solution is working. As each demo increases the orderer's understanding of the technical solution ideas and possibilities, deciding on what changes should be made (both long term and short term) for the specific functionality becomes easier. A few developers (Db, De) noted that performing demos more frequently has shortened the feedback loop with the orderer, e.g. helping to catch changes early. However, one of these developers (De) pointed out that doing demos does not assure that the orderer will not change his mind in later phases of the project. According to two developers (Db, Dg), demos might not be as useful for technical (non-visual) projects, in cases where the orderer is a product manager. One of the developers (Db) explained that this is because product managers require a graphical representation in order to fully understand the solution idea. However, apart from these less visual projects where doing demos might not be as successful, no interviewees reported on any negative experience with conducting demos.

Therefore, the practice can as such be recommended where the orderer can validate the demo through a GUI.

The impression of the authors is that Product Management plays an important part at the company in the big perspective. Product managers are responsible for prioritizing work items for their teams, as well as handling estimations regarding effort. These estimations of effort, although sometimes provided by the development organization, is gathered and analyzed by product managers in order to plan future work. Through balancing effort and value (prioritization), their task is effectively to maximize the future profits for the company using the resources (developers) that they currently have available.

5.2 The use of available sources for requirements knowledge

In traditional RE [59], a requirements specification is used in order to keep track of functionality and behavior of a product or a piece of software. However, as described in section 3.1 "The requirements context", the main part of the company's documentation that contains requirements is project specific and not maintained after the projects have ended. Therefore the employees cannot depend solely on requirements documentation, when trying to clarify what the correct behavior of the software is. According to previous research (presented in section 2.2.3 "The challenges of Agile Requirements Engineering"), the use of other sources is a common approach for finding requirements knowledge in situations where the requirements documentation is insufficient. For example, the research claims that code and test cases are often used for this purpose.

When asked about how the correct behavior of the software is uncovered at the company, all interviewees referred to several other sources than requirements documentation. Moreover, the views on what sources that constitute requirements differed greatly among the interviewees. For example, following a discussion about a document that previously was used to specify a project's purpose, a senior tester (Ta) stated:

> *"In some way this is also requirements documentation, but it is more like a fluffy idea of why we are doing this. Then this might yield some use cases on the problem that should be solved [by the project] and the use cases yield more detailed requirements that yield a design specification […] and in some way the line between what requirements are [starts to get blurred]."*
> - Ta

Due to the varying perception of what constitute requirements at the company, many sources containing requirements knowledge were identified during the interviews. These sources included documents and tests, as well as non-written knowledge. Since all of these are used across the company for conveying requirements knowledge, they are interesting from a requirements perspective. One developer (De) expressed the situation at the company like this:

> *"There is no perfect model. Axis has chosen not to have a formal requirements database in the same way that many other companies have. It is a bit more, so to say, fleeting. On the whole, I think it works well. If we were to document all the requirements, then we would have to keep them updated as well. That would mean more work for us. Currently, the requirements are on different levels; a project that develops new functionality and sets the requirements, sure, in that case there are a lot of discussions and you have to write things down in a good way to get all parties to understand what you mean. […] Then we have the maintenance work. In that case it can be hard for new employees, for example, like 'What are the requirements for this? Is it working correctly? How should it work?'. Then it can be hard to look up as we do not really have a requirements database. However, the way it works at Axis is that you talk a lot with each other to get that information. Because the information is there, but everything is not written down. Someone knows something about how it should work."*
> - De

In similarity to the above quote, the interviews revealed that developers talk to other employees in order to find out the requirements. Many of the developers (Da, Db, Dc, De, Df, Dj) claimed that simply asking colleagues when trying to uncover the expected functionality of the software is common. Some developers as well as a project manager (Da, Db, Dd, De, Dj, PjMa) also mentioned that requirements are refined during the projects through discussions in the team and with the orderer. The project manager (PjMa) expressed that this works well, since it facilitates knowledge sharing. However, after projects are completed, information about requirements is not always available in written form. Therefore, this information is rather conveyed through verbal communications.

A number of the developers (Da, De, Dj) said that they communicate verbally with both product managers and other developers. Whereas product managers have an overview of the functionality of the software, developers have more detailed knowledge of specific functionality, especially the developers with CBA or CBM responsibilities within the area in question. According to interviewees (Da, Dc, De, Dg, Dh) questions about specific areas are asked to these developers. Many of the

interviewed developers (Dc, De, Di, Dj) claimed they rather ask questions than read documentation. These developers thought asking questions about functionality gives good answers and works well. Also, they did not seem to mind having to answer questions. During the interviews one of the senior developers (Db) recognized that developers get quite a lot of questions. However, a majority of the developers that touched on the topic (Dc, De, Df, Di, Dj) argued that answering questions is not an issue for them. Instead, one of the developers (Df) expressed that the approach could even be beneficial from the large perspective:

> *"Answering questions is not a problem, not for me at least. Of course it takes time that could have been used for correcting bugs, but I think that more bugs are corrected in total if you help solving each other's problems."*
> - (Df)

Many developers mentioned that they also use other sources of information than verbal communication, when trying to find out the functionality of the software. Seven of these (Da, Db, De, Df, Dg, Dh, Di) claimed they read the code to gain understanding of the functionality. One of these (Da) also mentioned manually checking functionality in the software through trying it out on a product. Moreover, one developer (Df) as well as a line manager (with 2 years of experience in that role) that was interviewed in an introductory meeting argued that, in reality, the tests developed by QA constitute the requirements.

Several interviewees (Tb, Tc) from QA expressed that they think their test cases can be seen as a representation of the requirements. A senior tester (Ta) agitated for focusing on test cases instead of requirements documentation:

> *"I try to move us in a direction where we have some kind of test driven development, where requirements yield a product and the tests test the product. Then the tests become the living requirements. [Imagine] we create product one, we have done the tests for it and some tests fail. Then you discuss in the project and you don't think the test should fail but that the product is correct and through the discussion you, so to speak, change the requirement that the test is. When you then have released the product, meaning you have a functioning product and tests that pass, the tests are good documentation for how the product should work. […] instead of discussing a requirements document, that determines the tests, that determines the product, and... It only becomes overhead [costs]."*
> - Ta

The senior tester continued with describing how products also can be seen as requirements, further reducing the need of requirements documentation:

> *"I think that, on the [platform department], there are many people who complain that you say 'This product should be as the other one' or similar. But that is very good, to have some sort of benchmarking. All of the products that we have released are in some way a set of requirements. This is much better than a long list in a document that might not even reflect the actual product. A long list in a document that describes how someone, at some point, thought the product should be. But we have the product right here, we can just check how it works. It is released and people are buying it – we can simply test it. Therefore, I am an advocate for using tests and products as requirements and that these two should be enough, meaning no other [requirements] documentation than tests and products should exist."*
> - Ta

The methods used by QA in order to gain requirements knowledge are similar to the methods used by the developers. Several testers (Tb, Tc) said that they communicate with developers in order to gain knowledge about requirements. Furthermore, these testers mentioned that both the PRS and the PFD are used to find the requirements. However, as the PRS is quite often missing (or of low quality), one tester (Tb) claimed direct communication with developers is used for clarifying the requirements. The same tester also thought talking to the developer of the software always gives better answers than reading documentation. However, as the data sample from QA in this regard was not extensive, more research is needed in order to explore how other parts of QA work when uncovering the requirements and what they think about this specific approach.

Direct communication with other employees was also claimed (PdMa) to be used by product managers when they need to know how the software is working. Furthermore, the product manager claimed that simply trying the software out is used as a method for finding the functionality in the software. Another product manager (PdMb) summarized how requirements are found by developers when answering a question about how they can uncover the requirements in an unfamiliar module:

> *"The unit tests and function tests for that particular code. It is really that, and then they talk to the owner of that code in case the change they want to implement destroys something. [...] It is the tests and those end user documents like PFD and CD that describes how the behavior should be."*
> - PdMb

Overall, the direct communication with colleagues seems to be one of the most important sources for requirements knowledge. Moreover, many interviewees seemed content with using this type of communication for conveying requirements knowledge. However, this conclusion is restricted to the platform organization. The interview data suggested that there are challenges in other parts of the organization, but future research is needed in order to validate these. For this reason, they are not presented in detail here.

5.3 Quality requirements

When asked about quality requirements, many of the interviewees (Dc, Dd, Df, Dg, Di) intuitively related it to performance requirements. Because of this and the fact that quality requirements were not the main topic of this study, much of the discussion in this section is based on the context relating to performance requirements.

The orderer is the one who ultimately decides on quality requirements questions. According to Di, the orderers are sometimes aided by people with more technical knowledge such as the CBAs. However, putting quality requirements directly on the platform is not trivial as the performance between products varies greatly, e.g. due to different hardware as well as different functionality setups. One project manager (PjMa) mentioned that orderers occasionally need to change the initial quality requirements in a project, because they were not feasible.

Furthermore, one of the testers (Tb) explained that there have been issues with pushing more functionality into the platform without upgrading the corresponding hardware. This caused older products, with older hardware, to not be powerful enough to support the new software. Additionally, a senior tester (Ta) from QA mentioned that specifying quality requirements is dangerous, since the quality desired by the customers always keeps increasing. Thus, the tester reasoned that any old specifications that define “high quality” may actually represent low quality in current terms. The tester concluded that there is a risk that people will interpret those kinds of specifications as the absolute truth, without considering that quality needs to improve constantly. Therefore, the management of quality aspects, and thus quality requirements, is complex at the company due to several different reasons.

According to the view of one product manager (PdMa), there are no processes for dealing specifically with quality requirements, other than the quality tests. Rather than having clear guidelines for quality requirements, this product manager found quality aspects to be handled from case to case, based on the opinion of the related product manager. In order to be able to do this properly, interviewees (Di, Ta) indicated that product managers are dependent on the knowledge of others, e.g. the CBAs, in order to be able to answer questions about quality aspects. This communication might pose a challenge, as Ta notes:

"The one who ultimately decides [on quality requirements] is the orderer, the product manager, but they can't express themselves in a way that, in reality, puts them in charge. Instead, they are dependent on that people from the technical roles [e.g. developers] can explain the difference between alternative A and alternative B to them. There are often big communication problems here, between the people who aren't interested in the technical details and the people who only are focused on the technical details. They find talking to each other very difficult."
- Ta

Several developers (Df, Di) noted that there exists some confusion about quality requirements. In some cases, the confusion is about what the required level of quality is, while in other cases it is the reason for having certain levels of quality that is unclear. One developer (Df) reported on regular confusion regarding a specific quality aspect, where a test case had failed and it was unclear to the different parties what quality should actually be required. The developer explained that the failed test cases only led to decisions of ignoring them. Three developers (De, Dg, Dj), a majority of the ones who touched on the specific aspect, revealed that quality requirements usually are not specified explicitly in their projects.

5.3.1 How the evolution of quality aspects is managed

The company has started to measure performance in order to keep track of it over time. The measurements are taken both by the projects and by QA, as they do quality testing of the software. One senior product manager (PdMb) mentioned that this enables management of performance, as the measurements make it possible to follow the development of performance over time. The product manager claimed that having this kind of information for example makes it easier to see trends of degrading performance. Thus, measuring the actual performance does work as a way of managing quality instead of using written requirements. This means that performance aspects are measured after implementation in order to ensure that they are adequate. However, in order to draw any conclusions on whether this approach is appropriate in the context of the company or not, further investigation is needed. Nonetheless, considering the complexity of managing quality requirements in an SPLE setting where many different products use the same platform, this approach seems to be a reasonable option.

Several developers (De, Df) saw QA as the responsible party in checking the quality aspects and making sure they do not deteriorate. This view is similar to the ones of a tester in QA, who felt that QA in some way is made responsible for defining the quality requirements (Tb). This seems to be due to the fact that several members from

QA (Tb, Tc) thought there is a confusion regarding what is correct and not when it comes to quality aspects. As Tb noted:

> *"Even when we do not have any requirements we still have to test and show the quality of the products. We have to test the quality and give the reports to show how good it is. Others then look at it and decide if it is good. We become a part of the requirements, for better or worse. We can't just sit and wait to get the requirements – in that case not many test cases would have been run. We have to, sort of, invent the requirements and test cases anyways to get it [the software] through."*
>
> - Tb

Because of this, the same tester also felt restricted to measuring what the product "can" do rather than what it "should" do, expressing a concern that not having clear goals for quality requirements might make it hard to manage the direction they are heading in. Another tester (Tc) gave a similar explanation, also adding that QA has had problems knowing what quality levels that are considered sufficient. This tester felt it becomes time consuming for people in QA to sort these kinds of questions out. This indicates that the current approach of managing the quality aspects may be causing QA to compensate by spending time with specifying the actual quality levels in their test cases.

5.4 Benchmarking as requirements

Several interviewees (Tb, Tc) reported on a common usage of requirements relating to older products or to the software platform itself. These requirements are usually expressed in a comparative way, using the existing functionality and quality properties as a benchmark. In this report this concept is referred to as "benchmarking as requirements", which indicates that the requirement in itself is stating the desired quality in relation to some other existing software. Two of the interviewed testers (Tb, Tc) had many examples on this kind of requirements, e.g. "Product x should be as Product y, but better" and "The new version of the software must not be worse than the last version". As another example, one of the most common requirements of this kind is a requirement on backwards compatibility, which is put on a large part of the company's software. Since it is an important requirement for the company, one interviewee (Df) explained that when asking the question "how should it work?" to Product Management, one of the most common answers is "it should work as before". Benchmarking as requirements is encountered, at least to some extent, in development of code and test cases for new software.

The opinions of whether or not to use benchmarking as a form of requirements varied between the interviewees. One tester (Tc) emphasized that such requirements are

hard to test due to difficulties in understanding what is included in a piece of software, as some other software is used as comparison in the requirement. This difficulty, in turn, seemed to be one of the causes for difficulties in choosing test cases for a piece of software. These issues, relating to using benchmarking requirements, are explored more in section 5.5.2 “Understanding which functionality is included in a piece of software“, and section 5.5.4 “Choosing what test cases to run”, respectively.

Another challenge with using previous products and platform versions as an oracle for test cases was presented by this tester (Tc). The tester reasoned that there needs to be some process that guarantees that those products and platform versions are correct in order to be able to effectively do testing, as otherwise defects in old software will not be found in the new software either. The reason for this would be that just doing comparison between artifacts will only verify that the artifacts are alike, not that the new artifact that QA is testing does not have any defects. On the other hand, a senior tester (Ta) felt that these requirements are good from the perspective of Product Management, expressing that using this kind of comparison is much easier for a product manager than answering detailed technical questions:

> *“When you release [the product], then you have taken that decision. Then the performance of that product will become some kind of benchmark. After that it will be much easier for a product manager to say 'we are going to do product two and it should be at least as good as the first one'. Then we have that benchmark and we don't need documentation of what the first product could do, we can simply measure it.”*
> - Ta

Note that although the tester is speaking specifically about performance aspects, the logic can be applied to functionality and functional requirements as well. By making the decision to release a piece of new software or a new product, the senior tester reasoned that the quality of the product, including its software, should be seen as adequate for the market. The quality in this case can also be seen as the degree to which defects exists in the software. Thus, handling any defects that are left in subsequent products and their software might not be critical and could potentially be handled on a case-by-case basis.

As the effectiveness and efficiency of using benchmarking as requirements have not been specifically studied, further work is needed in order to conclude whether this approach is beneficial or not. However, the concept as such is interesting as it could reduce an organization's dependency on requirements documentation and thus possibly facilitate lighter processes. As no previous research was found on such a concept, these ideas may be interesting as future research material. As the concept has the po-

tential to reduce the costs of creating and maintaining requirements documentation, it could serve as a cost efficient, "good enough" way of specifying requirements.

5.5 Implications of the company's requirements process

In this section, the consequences of using the current requirements process in the company are presented. This includes a discussion about the following:

- 5.5.1 Understanding the intended behavior of a feature – presents the interviewees' viewpoints on any issues related to this topic.
- 5.5.2 Understanding which functionality is included in a piece of software – presents the interview results and some discussion about the significance of the challenges in regard to this topic.
- 5.5.3 Complementary information about requirements – presents what is to be regarded as complementary information, which of this information that could be useful to the interviewees and why.
- 5.5.4 Choosing what test cases to run – treats the difficulties, revealed in the interviews, of choosing test cases for a piece of software.

5.5.1 Understanding the intended behavior of a feature

Two interviewees (Df, Di), as well as a line manager (with 6 years of experience in the current role) that was interviewed in an introductory meeting, reported on trouble with knowing what the intended behavior of a feature is. One of the factors contributing to this might be the difficulty of getting an overview of the PFDs for the platform. This was expressed by one developer (Di) and one tester (Tc), where the tester specifically stated:

> *"The PFDs are located at the different project sites. […] It is really hard to get an overview of all the functionality. It is a big problem. […] It is hard to get an overview of the PFDs and from where they originate."*
> - Tc

Since the PFDs are used in order to understand the software, this issue can have an effect on the process of finding out the intended behavior of the functionality. More specifically, not being able to get an overview could contribute to not understanding the specific functionality, since the functionality of a certain piece of software generally is spread out between several PFDs.

A project manager and two developers (Df, Di, PjMa) also indicated that QA has some difficulty in knowing how certain functionality is supposed to work. PjMa and Df stated that it is not uncommon that developers get incorrect tickets. One of the

testers (Tc) gave a possible explanation to this, indicating that their test cases sometimes fail because the tests are inconsistent with the intended behavior. According to the tester, one of the reasons for this is that the behavior of the software is occasionally updated without notifying QA, causing test cases to be outdated.

Incorrect tickets happen even though the process for validating test cases is quite rigorous at the company, according to one tester (Tc). The tester explained that the company uses reviews, where employees from different departments attend (e.g. author of the PFD, other members of the project, the responsible QA resource), in order to uncover and solve inconsistencies between the intended behavior and the test cases. Ultimately, the inconsistencies lead to failing tests and thereby that the developers receive incorrect tickets. Since resolving incorrect tickets takes time for developers, this might cause inefficiency at the company. However, a senior tester (Ta) claimed that tickets are means of communication, explaining that instead of asking the developers directly, testers can send a ticket that might be wrong. According to the logic of the tester, it is not certain that incorrect tickets waste developers' time, since the alternative – of having to answer more questions from QA – could be just as time consuming. Therefore, reducing the number of incorrect tickets that are created in the organization might not be the most pressing matter in the current context, especially when considering the open culture at the company and the focus on communication.

5.5.2 Understanding which functionality is included in a piece of software

Some interviewees (PdMa, Tc) experienced a challenge in knowing the scope of the functionality in a specific piece of software. This piece of software can either be a specific version of the platform, a product specific piece of software, or in some cases customer specific service releases. The product manager (PdMa) expressed difficulties relating to this issue in the following way:

> *"This is one thing that I find pretty difficult today, to know what is actually included in a given software or product. […] You can always go back and see what was integrated into a certain LFP [version], but there is not one place, like a list, where you can find it. I think this is a deficiency, since you often need that information and to know if certain functionality was included in [version] 5.60 or [version] 5.55."*
> - PdMa

Similarly, one tester (Tc) explained that there is much confusion in this regard, especially when testing the so called benchmarking requirements (described in section 5.4 "Benchmarking as requirements"). The tester pointed to that such requirements are vague and hard to test, as it is hard to know what is included in the software that such requirements benchmark against:

"As a test engineer I would like the requirements to be verifiable and that is not always the case. As I said [referring to requirement], '[the product] should have all the functionality that exists on the platform'... Well – where is that specified?"
- Tc

No issues were reported from the developers on this topic. However, this problem might not be applicable for developers, as they tend to focus more on specific functionality rather than look for information on greater portions of functionality. Also, the interview data is somewhat inconclusive on the generality of the issue in Product Management and QA respectively. Further work may therefore be preferred in order to verify that these difficulties are indeed significant in QA and Product Management. Additionally, the difficulties might be eased from the ongoing improvements, which are elaborated more upon in section 5.8 "Ongoing improvements". However, the confusion regarding what is included in a piece of software causes additional issues. Specifically, the interview data indicates that the confusion causes difficulty when creating test suites (elaborated on in section 5.5.4 "Choosing what test cases to run").

5.5.3 Complementary information about requirements

In some scenarios the requirements that do exist may not contain all the information that is needed in order to be able to convey a deeper understanding of them. This complementary information may, for example, contain details about the importance of the requirement, who ordered it and what the rationale for specifying it in the first place was. This section explores what the interviewees expressed with regards to finding this kind of information.

A motivation for having this kind of information was given by a senior product manager (PdMb), who raised concerns with letting tests completely replace the requirements in the following way:

"One reason for wanting something that keeps track of the correct behavior is that when someone tries to correct an alleged bug, you want to know whether it really is a bug or if it was ordered to work that way. Otherwise, there is a risk of drifting away from the initial idea through QA, where QA in some way drive the requirements [through their test cases]. QA might think something doesn't work well enough for the customer, while the project and the orderer might have an agreement that a certain feature is not super important. […] For this reason you want some kind of documentation claiming the purpose [of the feature], why it was done, how important it is and what was actually agreed upon."
- PdMb

Regarding the purpose of a feature, many of the interviewees (Da, Db, Dj, PdMb) expressed that it is difficult to know why a feature exists, how important it is or why it was chosen to behave the way it does. One of the developers (Dj) elaborated on the difficulties in finding out why certain decisions relating to functionality has been taken:

> *"This is something that we have been struggling quite much with. […] For example, you can set one of these overlay texts where you choose the color of the text to either be white or black. Then you can also choose the color of the background, which can be white, semi-transparent white, black or semi-transparent black. The thing that happens when you put black text on black background, well you understand that... It gets black. […] This example is maybe not that severe. It is nothing that we are depending on. But I have experienced other cases where we have had a bigger need of knowing the history behind the decisions."*
> - Dj

One of the product managers (PdMb) mentioned another issue due the difficulty in knowing the purpose of certain functionality, when asked if it was possible to find out why a feature behaves in a certain way:

> *"Not always. It can actually be pretty hard. If there is a CD you should be able to find it in there. But it could be that you find why it works as it does on some level, but not why it was interesting to do to begin with and what the reason was that you developed it. That can be pretty hard to find out."*
> - PdMb

When asked if the CD did not contain a motivation the product manager replied:

> *"Yes. It can describe something like 'In order to support this use case it needs to be like this', but it might not be documented from where the use case originates. So if it [the requirement] turns out to be very expensive, error prone, difficult to maintain or hard to test and you want to remove it, it can be hard to know if that is okay since you have no idea of what part of the market that uses it and who actually ordered it to begin with."*
> - PdMb

This indicates that the product manager thought it would be beneficial to have some sort of complementary information together with the requirements, specifying their stakeholders and their purpose. However, according to the product manager this

would probably not be viable to document this since the cost of implementing such a solution would likely outweigh its benefits.

5.5.4 Choosing what test cases to run

Although this report does not specifically treat product development challenges, some product specific challenges from the testing perspective are presented in this section. The reason for elaborating specifically on these challenges in relation to testing, is that test cases have been established as a fundamental aspect of how requirements knowledge is handled at the company. As the tests, to an extent, act as requirements at the company, these challenges are important from a requirements perspective.

The difficulty of choosing what test cases to run for a specific piece of software was reported on by several testers (Tb, Tc). As the company is conducting SPLE, developing against a platform, testing is conducted in several places. For example, tests are run when a project integrates its work to the platform, when a new platform version is about to be released for a product group and when a new product (with a certain functionality setup) is developed. In all of these cases, QA staff needs to make an assessment of how many tests to run and which test cases are applicable.

Even if tests were considered as requirements in some ways, the tests are not equivalent to requirements. One tester (Tc) was uncertain of how to ensure proper test coverage and complained about the lack of traceability between tests and requirements. However, because of the incomprehensiveness in the company's requirements documentation, having traceability between tests and requirements might not solve all issues. Something that could be done instead is to create traceability between tests and products, making it easier to know what test cases to run for each product. The tester (Tc) recognized that a mapping between requirements and products essentially would mean the same work as doing a mapping between tests and products. As the company's requirements documentation would need significant work in order to become comprehensive, the viability of the second option could be worth evaluating.

The same tester showed general reluctance to using tests as requirements, expressing the following concerns:

> *"As long as you save input from other test rounds it could work. But it easily gets very messy when [the development of] a product is divided between different projects, when it is an umbrella project and test rounds are split up. It is hard to know... Okay, we should run all test cases that are applicable on this product, but where do I start? […] The problem is that it is very product specific."*
> - Tc

In essence, the tester was pointing to difficulties in finding what functionality is supported in each product (similar to section 5.5.2 “Understanding which functionality is included in a piece of software”), as well as a difficulty in finding old test data. Old test data are specifically used when making test runs for new LFP versions and can thus help a tester in choosing the relevant tests. This gets critical when having requirements that use the benchmarking concept, as the tester in many cases is dependent on the old test data in order to know what tests to run for any new software.

To summarize, knowing what test cases should be run is made more challenging through low traceability between tests and requirements/products, difficulty in knowing what functionality that is supported in a piece of software and relatively low access to old test data. As the tester (Tc) puts it:

> *“Often, very many tests are not applicable since you cannot tell that they are not applicable when you choose [what test cases to run for a product]. You rather choose too many [test cases] than too few. This means quite much time is spent later by the tester trying to sort out issues like 'This doesn't seem to work', when in fact the test case is not applicable.”*
> - Tc

Thus, the interview with Tc indicates that testing is made significantly more cumbersome through the absence of rigorous requirements documentation. However, the ongoing improvements (see section 5.8) that are currently in progress at the company might at the same time offset the challenges. Specifically, the definition of features that are being collected in a comprehensive feature list could have a big impact on the testing processes. Therefore, future evaluation of the effects of these improvements in a testing context would be useful to complement the findings in this study.

5.6 A remark on the general quality of the software

Although the above sections have treated a number of challenges, the general performance of the organization as a whole seems to be satisfactory from several different parties' point of view. For example, one developer in Product Maintenance and one tester (Dc, Tb) noted that the quality of the software at release is good, in some cases compared to other companies where the interviewees had worked previously. Especially, the developer (Dc) working in Product Maintenance, the department responsible for development specifically aimed at customer issues, expressed that the customers in general are happy with the quality of the products. These statements, together with the fact that the company is the market leader in its area, points to that the general quality of the software that is being released by the company is relatively good.

5.7 The scalability of the requirements process

The majority of the interviewees that had an opinion on scalability (Db, De, Dh, PdMa) were not concerned of scalability with regards to the requirements. A few interviewees (Di, Tc) recognized scalability as an issue in certain contexts. Still, none of these targeted the process in general. Instead, Tc was worried that the use of benchmarking as requirements would not be sustainable, whilst Di expressed scalability concerns in his/her own team where documentation had been neglected due to other priorities.

More specifically, one of the developers (Dj) did not think more documentation would make the process more scalable. Also, a product manager (PdMa) related scalable processes to being simple and light (including documentation):

> *"The risk is always that you paint yourself into a corner somehow, generally with regards to too much documentation. There could maybe be a risk that you get to a point where it is all about a documentation process, that the process is important, rather than developing good functionality. That feels wrong to me, I would rather have it the other way. That risk is probably always there, but I don't see any short term tendencies for it although it is absolutely something to keep track of. This was something we saw a bit of in the [old version of] CD, which I mentioned, that we wanted to have everything there. […] It became very cumbersome already from the start, and we abandoned that idea. I think you need to avoid these kinds of things to keep the scalability. […] If you get this monolithic document, then the scalability goes really bad. […] It is always a challenge, like 'okay, how do we split it up', it is difficult."*
>
> - PdMa

At the same time, the product manager recognized that it is hard to have a single process that suits many teams, as different teams tend to work in somewhat different ways. The product manager reasoned that because of that, there is a risk that all elements in a large process will not be needed for all teams.

On the other hand, one developer (Dg) pointed out that the process is subject to constant change, not least when considering the pace at which the company is growing. The developer explained the development of the company's process with regards to scalability:

> *"This work [with more structure and documentation] is something that has been started now and is in progress now, because we are growing. The PADs and so on have been added because we have grown. It worked without PADs before when the company had 100 persons or something, but a*

couple of years ago we suddenly became 600 and then it started getting harder. Then these documents were added, as a result of that things got more complex. […] The process is adjusted to the company."
- Dg

Finally, a senior tester (Ta) expressed a viewpoint against large processes and emphasized that the people in the organization ideally should take a large part of the overall responsibility:

"When you get more and more overview, you lose more and more understanding of the details. You have to try explaining to people on a high level. After that, people have to decide for themselves. I do not think that it necessarily is good that there is one person in the top of the building, deciding what is good and bad, right and wrong. It becomes an extreme bottle neck. You can see it a bit like Wikipedia. There are no editors. There are some rules about what is good and bad, and there are people who look around and point out that 'this page is not well written' and things like that. But for the most part, this is distributed to the masses – people go in and edit, write and decide for themselves what is right and wrong. That scales so much better than large processes and reviews – if you were to have reviews on all Wikipedia articles before they were published then nobody would bother with that."
- Ta

According to a senior product manager (PdMb), scalability becomes an issue when more people have to work on many components as the platform is growing. However, the company is currently modularizing the platform in order to take care of challenges related to scalability. Following a question on the amount of questions the developers have to answer the same product manager mentioned:

"I don't know if they [the developers] really have to ask that much. I hope people talk all the time in their teams, but hopefully they don't need to talk that much between the teams. That is what we are working on heavily right now and for the future. That we should modularize more and have clean APIs between things so the communication channels within the platform are clear and as few as possible. Because, as soon as you need to talk to someone in another functional area it means you have to use their code in some way, which you should be able to do, but you should do it as seldom as possible. Each such thing creates a dependency within the platform."
- PdMb

The statement shows that the company applies architectural measures to reduce the amount of questions that are asked between members of different function teams. These questions can for example be about requirements and functionality in different functional areas. The reduction of the number of questions could mean that each function team becomes more independent, giving them the ability to focus on their own area without having to look into other areas. The teams may use APIs created by others, but should preferably not have to touch their code. According to another product manager (PdMa), the architectural efforts in this regard will also reduce the need for close communication between the product managers, which currently is needed due to challenges with managing quality requirements.

Moreover, by splitting up the platform into smaller and independent modules, requirements and the functionality will be split up into smaller portions. This means that each function team has responsibility of a small piece of code together with its corresponding requirements and functionality. The authors' interpreration is that this will make it easier for the team members to keep track of the requirements, reducing the need for documentation. Also, it will increase the degree of specialization within each team, making the team members more knowledgeable, e.g. about requirements and functionality, in their certain area.

However, if the functional areas grow, so will the teams and more people will have to work on the same code. A bigger team would mean that a bigger piece of code constitutes the functional area. The authors' interpretation in this case is that a larger team will make it harder for team members to keep track of the requirements and functionality, simply because the code base (and thus the amount of requirements) is larger. This results in an increase in the amount of questions that needs to be asked about requirement related topics. Following a question about how the company managed organizational growth a senior product manager (PdMb) replied:

> *"If we have a function team on the verge of being split up where the problem is that the code built in a way that makes it difficult to split up […], we put it as a task in the roadmap. Just because the teams have to be able to grow. A team cannot grow indefinitely. If it grows to be more than 6-10 members, depending on the complexity of the area, it will become hard to communicate and keep track of what other team members are doing [...]. Then you have to split up the team and in order to make it worthwhile you also have to split up the code."*
> - PdMb

Based on the above elaborations on the topic, the interviewees did not regard the requirements process as such to cause scalability issues. The general concerns were rather, as one interviewee (PdMa) suggested, that the process and documentation must

not be allowed to grow too much and become cumbersome. Essentially the interviewee expressed a desire to keep the process as lightweight as possible in order to mitigate any scalability risks. Also, the company appears to be paying attention to risks associated to scalability and is continuously evolving its processes in order to make them more scalable, e.g. through architectural means such as modularization.

5.8 Ongoing improvements

The company has several parallel ongoing improvements in progress aiming to improve their processes, some of which also affect the requirements processes. The different improvements are discussed with regards to what benefits the initiatives might yield from a requirements perspective. Based on the content presented further down in this section, many of the improvements that are elaborated on seem to relate to, or even originate from, the QA department. This might point to that testing has significant impact on the requirements process within the company. It might also relate to that QA seemingly experience the most significant issues with the current requirements process (see section 5.5 "Implications of the company's requirements process"). Of the improvements found, the most important ones are explained in detail further down in this section, but summarized here:

- Implementation of automated tests that will shorten the developers' time to feedback on their code.
- Auto-generation of feature lists that specifies the software functionality in the platform and in the products.

Promoting automated tests, a senior tester (Ta) reported on long feedback cycles for developers as well as finding faults too late. This has caused the company to push testing "upwards" through the development process. From the developers' perspective, testing in QA has taken relatively long time. As the tester put it:

> *"If we see it from a time perspective, the typical scenario has been that you have an idea about what you want to do and do a requirements specification or something. And then the developers start, and they do an alpha or something. Finally they do a beta, a considerable amount of time later and now we start testing, roughly. Here the quality is really bad and at this point we start finding faults. This process might have taken weeks. It becomes very problematic for this project to not find the faults earlier. Some faults might come from other projects, finished earlier, not caused by the current project. [...] So, when this project finds a defect, it can be due to something that someone else did several months ago. Then it gets very diffi-*

cult to get ahold of the person who worked with this thing and now he doesn't remember anything. That the feedback loops become so long is a problem. What we have tried to do is partly to push the testing upwards, so that you discover as much as possible as early as possible."
- Ta

As the tester recognized, long feedback loops are an issue – the developers need to know if their work satisfies the requirements. When the requirements cannot be found explicitly, tests are one way of giving the developers this feedback. This is also recognized by a developer (Dg), expressing a wish for automatic testing of the requirements in order to get earlier feedback. Thus, minimizing the time between development and testing is important from several aspects.

One fact that influences the interaction between development and QA testing, is that test cases run in QA traditionally have been focusing on system testing from a black-box perspective. A senior tester (Ta) explained that these kinds of tests only are a nuisance to developers, as they only let the developers know "that they are not finished yet". The tester reasoned that it is difficult for a developer to have to wait several weeks just to get feedback on the code.

Furthermore, the tester elaborated on ongoing work in tackling this through automated tests, which could shorten the feedback loops:

"What I have been working a lot with is making [automated] test cases that the developers can run themselves, on their desktops, and not having to wait on building a firmware to QA where someone has to test it. Instead, the developers should be able to test their work themselves in order to get these shorter feedback loops. […] What is problematic with the current way of working is that if we have a lot of manual testing then that does not scale very well. It becomes very expensive to add an extra test run. The advantage of automated tests is that they only are expensive to create, while they are very cheap to execute. It is the other way around with manual tests, they are cheap to produce […], but very expensive to execute."
- Ta

Except the automated tests, QA is also one of the key parties in developing a more precise definition of the features that exist in the firmware platform. The idea is to specify all the features that currently exist and then maintain this comprehensive list of features. The tester (Ta) explained that a feature will likely be defined by a few use cases and corresponding test cases. Extending this to the current products, the idea is to enable extraction of product functionality through the product itself. Essentially, this means that it will be possible to understand which of the features that each prod-

uct has directly through the product. It is interesting to note that this is in line with the previous reasoning that a finished product should be a significant part of its own set of requirements. There are several benefits of defining features extractable through the product:

- It will be easier to keep track of the differences between products, as the extracted feature lists can be easily compared between different products. Implementing traceability between the features and the available test cases would also enable testers to find the test cases they need to run in a much more straight-forward fashion. This will at least partially solve the issue of choosing test cases. The defined features would also make benchmarking as requirements more viable, as a senior tester (Ta) notes:

  "We have a lot of products that have been released to the market. It doesn't matter what features they should have had, they have the ones they have. We have done scripts that enable generation of XML files that lists what features they [the products] have. Then, the orderer can say 'Well, I want a new product that is roughly like this one, but we also want this thing'. Again, you can use benchmarking towards that product [...] and see what features it has. The idea is that this feature list should be a kind of requirements document that you can start with."
  - Ta

- It will be easier to inform users and stakeholders of the supported use of the company's products. Through the use cases defining the different features, intended use of the products can be specified. Then, as the tester (Ta) and one product manager (PdMa) noted, these use cases can be used as information and possibly as a way of justifying changes affecting unintended use (as the unintended use is not supported):

  "I would like a couple of use cases to be defined, [specifying] how we think a feature should be used. Often [stakeholders] that come to you have done something outside [the intended use] and it is very hard to answer [those questions]. […] The important thing is that we document the cases that we have intended, because that is also something that we can use outside our organization – saying 'This is how we think that you will use this functionality. If you go outside of this in some special case, you are kind of on your own'."
  - PdMa

"In the future we hope that there are clear use cases. What will happen is that you will add a feature to the feature list […]. Then there will be a test, testing this particular use case making it kind of test driven. This test will define what the feature can do. It might be that you can do something else with this feature as well, but this will not be officially supported [by the company] and then we can break that later. We just have to warn everyone that 'this is how the feature should be used'. For all the old things this does not exist. For all the old things there is just a heap of stuff that can be done and the combination of what our customers can do is very, very large."
- Ta

- If the different versions of the feature lists correspond to the different versions of the LFP, it would be possible to get an overview of what functionality is contained in each LFP. The feature lists would thereby, at some level, give a partial solution to the difficulties in knowing what functionality is included in a specific LFP version.

However, one tester (Tc) notes a risk with using these kinds of automatically generated feature lists on products:

"[Following a discussion about the automatically generated feature lists]: For this to work there needs to be a process so that someone checks if it [the product's configuration] is correct. If we use the software in the camera as oracle we will not be able to discover faults that have slipped through [in the configuration]."
- Tester (Tc)

Although this risk is significant, the impression of the authors is that it might be mitigated if proper measures are taken. If released products are considered good enough to sell to the market, they should be good enough to use in benchmarking through their feature lists. Any bugs in the old product that need to be fixed in a new product can be brought up and taken into account explicitly if they need to. Thus, using released products as a benchmark from which the feature lists are compared does not seem to be tied to any larger challenges. On the other hand, if a new product's feature list is used in order to choose test cases, then the risk presented by the tester is a reality. In order to avoid this, the feature list in the new product must somehow be verified. One way of verifying this is to actually compare it with a product in a benchmark, making any differences between the products explicit and then investigate if these differences are the intended ones. Another way is to have someone with

extensive knowledge of the customer requirements verify the correctness of the new product's feature list directly.

Anyhow, even though it might be positive from many points of view, the feature list means one more step in the testing process where things can go wrong. Therefore, the process relating to the use of the feature list needs to be intentionally designed to mitigate these risks.

The company is also making an effort to collect data from test runs for future reference. This will to some extent facilitate the use of requirements in benchmarking form, as the history of test runs can be used as information on what tests should be run. However, one tester (Tc) still questioned the viability of this approach, as test scopes in many cases are split up between different products in order to reduce the time and cost of testing. As this means that any historical test scope will not reflect the full test scope, more rigorous collection of test data might not solve the underlying difficulties in choosing what test cases to run.

The improvements presented here will naturally have some effect on the benefits and challenges presented elsewhere in this study. However, due to uncertainties in the details of how the improvements will be implemented, this report will not extensively discuss the implications of these improvements.

5.9 Communication channels

This section treats face-to-face communication and documentation as communication channels, including how and when the communication forms are used and what the employees think of them. Especially, the purpose of documentation as a form of communication and as a facilitator for communication is considered. Note that all references to “documentation” in this section refer to requirements related documentation.

5.9.1 Face-to-face communication

The general viewpoint among the interviewees (Dc, De, Dg, Di, Ta, Tb) leaned towards a will to minimize the requirements documentation and instead focus on face-to-face communication. For example, several interviewees (Dc, Di, Tb) as well as a line manager (with 6 years of experience in the current role) that was interviewed in an introductory meeting stated that asking questions is easier and more effective than reading documentation. Some interviewees (De, PjMa) also pointed out that face-to-face communication leads to positive side-effects such as networking and knowledge spreading. Three developers (Dc, De, Di) preferred to ask questions rather than to read documents, when they needed to get an understanding of the software.

Face-to-face communication has been shown to be more efficient than documentation in certain contexts and is especially advocated in agile software development. At

the same time, research has shown, see section 2.2.4 “Traditional Requirements Engineering”, that face-to-face communication may not always be the best alternative. For example, if someone gets interrupted in their work by a question, the interruption will likely make the person lose focus and thus productivity is lost. However, not many downsides with face-to-face communication were brought up by the interviewees.

One of the downsides was expressed by a tester (Tc), who pointed out that even though face-to-face communication works well for one particular question, the answer is not saved anywhere. Therefore, another person who needs to know the same thing has to ask the same question. An increase in the number of questions on a particular area was also the reason that one team has started to build their own knowledge base, which they refer others to. To summarize, even though there is no general indication (with an exception in one team, see section 5.9.2 below) of that people tend to ask the same questions repeatedly, the problem does still exist in some cases. At the same time, the fact that one team handled an increase in questions asked to them through building a knowledge base shows capability of adapting to new conditions on team level. Also, it indicates a certain amount of self-organization within the teams.

5.9.2 Documentation

One senior product manager (PdMb) and one senior tester (Ta) indicated that one of the main purposes of documentation is to synchronize different parties, e.g. the different members in a team or a project and its stakeholders. The tester referred to this concept as “handshaking”. The “handshake”, even if not set in stone, serves as a common viewpoint of what a project should do. The product manager (PdMb) expressed his/her view on documentation relating to requirements in the following way:

> *“My take on [written] requirements is really that you never will be able to write a requirements list, which someone can take over and simply execute. Additionally, you will not be able to make the software execute correctly if you do not do any requirements either and just let them [the project] go. The communication is very important. […] One thing that you need is a way to make people talk to each other, whether that is through a document or regularly forcing them into a room – different organizations can solve that in their own way. You need to get them [the project and its orderer] to communicate at the right times. […]. We have chosen to communicate around the CD and PFD level, because that is where we found that the interface between our orderers and our project members was. In other organizations you may have more market oriented people participating in the projects and the orderer might be even more strategic, then maybe the communication with the orderer is taken care of by the market oriented people in the project. What level you want to have there is different between organizations.”*

- PdMb

The view was elaborated on by a senior tester (Ta), who presented a more radical view on documentation:

> *"As I see it, we have this flora of documentation. I think that all of those documents are very good in order to get people roughly synchronized. A test plan, for example, is very good because it is a kind of statement from the tester: 'Now I have read the requirements, now I understand what this project is about. This is my understanding'. Then people can read the test plan and say: 'Yes, that sounds reasonable' or they can say 'Oh, you seem to have misunderstood that completely'. It is sort of a handshake document. After that, I really think such documents are completely useless, but people want to go back to them and maintain them and... No, it will only get cumbersome to maintain – it is a handshake. After people have synchronized, they do the work through some kind of internal communication. If the project takes much longer than expected or if it changes very much, then maybe you do a second version of the document as a new handshake. Then you can have a meeting where you review that again. But, I see this kind of documentation as one, temporary, communication tool and not as the absolute truth which you should follow blindly."*
> - Ta

Some interviewees (Da, Db, Dh, PjMa) claimed certain documents are not used, to a great extent, after their creation. This further substantiates the view of documentation as a facilitator of discussion and as a handshake. As an example, the SWO was found to relate much to the concept of handshaking. On this topic, a senior developer (Db) stated the following:

> *"It [the SWO] is useful since you review it and think things through [early in the project]. But then when it is saved somewhere, I don't think that is very meaningful. Instead, its use is really that you have something to focus on during discussions."*
> - Db

No data pointed to differing viewpoints or other problems that indicate a lack of handshaking activities in the projects. Rather, one senior product manager (PdMb) expressed that the result of the software projects generally is satisfactory. This points to that the company's requirements process is adequate in its current form, with regards to handshaking and further communication of requirements.

Documentation was also found to have another purpose, namely as a way to store information over time, i.e. reference documentation. Relating to this purpose, some issues were discovered. However, the issues were much concentrated to one team where the documentation was especially lacking. In this team, it seemed that the number of similar questions were too many to be considered as optimal. A project manager (PjMa) gave the following answer when asked if it is a problem that people ask questions:

> *"No, I don't think so. It doesn't disturb me very much, but I think that the developers sometimes get a bit disturbed. It takes up a lot of their time.*
>
> *Q: Are there any questions that you feel the need to document?*
>
> *Yea... 'How does [specific component] work?', for example. A lot of these kinds of questions, actually, that we in [function team] feel we need to document in some way to get rid of all these questions. […] Some things might not even be related to [functional area], but it is rather things like you really should get in your introduction as a developer."*
> - PjMa

A developer (Di) from the same team had a similar viewpoint, expressing how the lack of documentation has caused introductions of new employees to take more time:

> *"It is hard for new employees and people outside [function team] to grasp how the component works. […] It takes time from us [to answer questions], also from within the team. I mean, everyone is new at some point and I think it has taken a long time to get into the work, partly because of this."*
> - Di

Additionally, one issue was reported from QA, where a tester (Tc) pointed out that occasionally there are no PFDs to be found for some functionality. The tester explained the consequences of missing PFDs in the following way:

> *"In the worst case we miss that new functionality has been included. This has happened. That we in a late stage of testing discover that 'This slider has not been here in any other product'. Then you have to look if there are any test cases for the functionality. If not, someone has missed it. It is either the project you are part of or it is another project where the functionality has been included and you have missed to write test cases for it. This means the functionality is completely untested by us."*
> - Tc

Both the above examples of issues relate to a lack of documentation compared to the level of documentation that the processes at the company actually demand. Thus, it seems that a minimal amount of reference documentation is important. In these cases, lack of documentation has led to a cumbersome amount of questions for the developers and the risk of missing functionality in QA. This means that even if six interviewees (Dc, De, Dg, Di, Ta, Tb) leaned towards a will to minimize the amount of documentation and instead emphasized face-to-face communication, documentation at some level seems warranted. At the same time, no developers outside the team in the above example experienced a lack of documentation as a significant problem. This indicates that ensuring the creation of the documentation that is required by the development process could be a reasonable starting point for all teams. This was further elaborated on by a senior product manager (PdMb), who implied that the process is adequate but that people do not follow it in all cases:

> *"I think that we have gotten everything down, process wise, on an okay level, but I don't think that we really get to that level in reality, yet. Therefore there are things to do, but we do not need to decide on more things to document. […] People forget to change [reference documentation] when they change some behavior and so on. It could be that no tests are written even though functionality is added and, in the next round, that functionality breaks without anyone noticing since there were no tests on it."*
> - PdMb

Thus, documentation in the company has been found to have two primary purposes. Firstly, it acts as reference for storing current knowledge for future use, e.g. as common ground when solving conflicts. The issue of similar questions being asked frequently, presented in the previous section, is another reason why an organization may want to have a certain amount of documentation. Secondly, documentation acts as a facilitator for discussion and as a way of aligning different viewpoints in projects.

5.10 Soft factors

In this section, human and cultural aspects are brought up and analyzed. Specifically, the introduction of new employees into technical roles such as testing and development is treated, as well as how knowledge of software is shared and what understanding the big picture means at the company. Through these sections, the company's requirements process is viewed from a new perspective, centered on the people in the organization.

5.10.1 New employees

The company's process of introducing new employees is based on mentoring (De, Dh). This means that the newly employed are assigned to mentors who guide them and answer questions when uncertainties arise. Also, the new employees are given simpler tasks in order to gain an understanding for their specific role and area. For developers, this means starting off by doing maintenance work (Dd), through which they can gradually increase their understanding of the code blocks in their functional area.

As previously mentioned (see the second quote in section 5.2 "The use of available sources for requirements knowledge"), some difficulties arise for newly employed during maintenance work due to the lack of a requirements database. However, according to the quote, this is solved through direct communication instead. One of the testers (Tc) raised the following concern with replacing the requirements with communication when claiming:

> *"As a [role] I try to forward [the new employee] through the existing information channels. A lot of [the process] involves talking to people and of course it takes some time before you know whom to ask. So much of the communication goes through us [role], meaning it takes time for me."*
> - Tc

The statement indicates that it is difficult for new employees to know whom to talk to. Thus, a lot of the questions are directed toward more experienced people that can forward the newly employed to people that might hold the answers. This takes time from the experienced employees, causing them to be less efficient in their work. According to previous research (see section 2.2.3 "The challenges of Agile Requirements Engineering"), new employees ask questions because of the lack of documentation. Thus, a more rigorous requirements approach could potentially facilitate the introduction of new employees, as it could alleviate some of the dependence on the more experienced employees. However, it does not seem like newly employed struggle to any greater extent because of the absence of rigorous requirements documentation. Instead, the three most recently employed developers (each with an experience of less than 2 years at the company) that were interviewed (Dd, Dh, Dj), claimed it is rather the process steps (e.g. integration of code), tools or architectural documentation that create difficulties for new employees. Therefore, difficulties for new employees in understanding the requirements for the software should perhaps not be the primary focus for the company when trying to improve the introduction of new employees.

5.10.2 Knowledge sharing

As mentioned in section 5.2 "The use of available sources for requirements knowledge", the company allocates responsibility, and thereby knowledge, between a

number of different roles. In essence, this means that QA and Product Management are responsible for having an overview of the functionality in the software, whilst the developers are responsible for keeping track of how their code functions on a more detailed level. This also includes requirements knowledge, which in many cases is held by the employees rather than written down in documents. Thus, the company puts a large part of the responsibility of keeping and sharing requirements knowledge directly on its employees. In this context, individuals with extensive knowledge are more important. Three developers (Da, Dg, Dh) indicated that the CBAs are included in this group of individuals, since they have overall responsibility and knowledge of the functional areas. Therefore, the CBAs are seen as important coordinators and information sources.

Two of these developers (Da, Dg) identified risks in that a lot of the requirements knowledge at the company is individual, especially when specific knowledge is only held by single employees. According to one of the developers (Da) there is a risk that if a CBA quits, knowledge could be lost on previous decisions and why the code functions (and if it should function) the way it does. Thus, it seems like a lot of knowledge, e.g. about requirements, is individual and undocumented. This increases the dependence on individuals and is probably one of the reasons that explain the high amount of questions asked at the company. The other developer (Dg) claimed that loss of experienced CBMs is undesirable, since code areas have to be delegated to other developers that might lack detailed knowledge of them. In these cases, it is not certain that the new responsible CBM can answer questions about his/her code and its functionality.

However, a few interviewees (De, Df) did not see the dependence on individuals as a major risk for the company, since there are always other sources that can be used in order to uncover the requirements. One of the developers (De) gave the following explanation to why unwritten requirements are not a problem for the company:

> *"Q: But what if someone leaves? Will unwritten requirements not be a problem then?*
>
> *No, I don't think so. In that case, you have to go to the code and see how it is implemented. If there are no apparent bugs, defects, then you can start questioning the functionality. How should it work? Then you go talk to Product Management who probably ordered it from the beginning, talk to the System Architect to see what [he/she] thinks about how it should work. Bring it up with stakeholders really, the ones who are affected, and try to sort it out – what is a sensible behavior in this situation?"*
>
> - De

One factor that facilitates the use of this approach is that the personnel turnover at the company in general is low, as explained in section 5.1. The authors' interpretation is that this means that individual requirements knowledge is kept at the company to a greater extent than companies with a higher turnover. Moreover, the company does not only have CBAs and CBMs for the different functional areas, but also backup CBAs and backup CBMs, thus spreading the responsibility and reducing the dependency on single individuals.

Another way the company reduces the dependency on individuals is through the function teams. Several developers and one project manager (De, Dg, PjMa) claimed the introduction of function teams has facilitated knowledge sharing. This includes requirements knowledge that is spread within the team during the break down of more detailed requirements, an activity performed together with the orderer. An example of this is that one of the developers (Dc) argued that if a CBM is unavailable, questions can be asked to the backup CBM or other members of the team. Therefore, it seems like the company limits the dependence on individuals and mitigates the risk associated to the loss of experienced employees.

Additionally, the dedication of QA resources to projects seem to contribute to better knowledge sharing and communication between the development teams and QA. This was substantiated by several of the developers (De, Di), claiming that QA members' participation in team activities makes QA more aware of what to test and how the software is supposed to behave i.e. requirements knowledge. For example, this means that differences in opinions regarding how the software should work can be sorted out earlier. Also, according to two developers and one project manager (De, Dg, PjMa) this participation leads to positive side effects such as increased understanding of the other department's work, both for QA and for the developers of the platform.

Several testers (Tb, Tc) shared the view that dedicating QA members to the development teams has improved the communication between QA and development. One of these testers (Tb) explained that the improved communication has resulted in better test cases, without introducing additional documentation. Also, the fact that the QA members review the teams' output and the teams review the QA members' test cases means inconsistencies and misinterpretations can be found earlier, thus potentially reducing the amount of rework.

5.10.3 Understanding the big picture

It seems that the company puts emphasis on software engineers' understanding of the wider perspective of the company's activities, e.g. business model, other departments etc. This emphasis is substantiated by a senior tester (Ta), who explained the

importance of understanding the big picture, as well as how the company promotes employees who manage to do it:

> *"If I go in and say 'This is how you must do it!', then they will probably listen to that. The next time they are going to do something, they will wait for me to tell them how – and then we're there again. It is really important that everyone understands this context of why something is done and so on. Therefore, it might be good that the requirements are quite bad, so that the developers and testers are forced to understand what Product Management is after.*
>
> *There is one thing that is very good at Axis and a reason for why this [requirements process] has been working at all. That thing is the career ladder for engineers and how the assessment is made of whether an engineer can go from Engineer, to Experienced, to Senior, to Expert. The assessment is only based to less than 50 % on the technical competency, the rest is about what understanding you have for Axis' business model, for other parts of the organization, how good you are at communicating, etc. etc."*
> - Ta

The senior tester went on to explain how this impacts the cooperation between engineers and Product Management, giving an explanation of how the wider understanding actually comes to use in the organization:

> *"If everyone understands those things, then it becomes much easier for Product Management to explain what they want to the engineers. In that case, engineers can anticipate what the requirements will be: 'I know that [he/she] wants it to be like this' or 'The product manager have not mentioned this, maybe [he/she] forgot it, but I know that this will be a problem for our customers'. Again, this is a part of Axis culture where you want people to Act as One and so on. That is what has made it possible for us to make such good products even though we have quite bad requirements."*
> - Ta

This indicates that one important factor that allows the company to use the current requirements process is the company's culture. Specifically, the requirements process seems to be facilitated by the concept of rewarding engineers who have an understanding of the bigger picture. One of the product managers (PdMa) also indicated that this understanding is desired:

> *"I have to watch out. I do not want to tell the members of a development team exactly what to do. I want them to think for themselves and see the*

customer's perspective. A large part of my job in this case is to facilitate a discussion where a number of engineers from a project are gathered together with the supposed users, and to get them to talk to each other."
- PdMa

However, even with the emphasis on having a broad understanding of the company and its customers, everything is not clear to all developers. One example is the purpose of the documentation that is written during the projects. A senior product manager (PdMb) expressed the following:

"Many of the documents have uses that perhaps not everyone in the projects is completely aware of. This causes the rationale that 'We don't have to write this, we don't need it'. Then someone else comes and says [for example] 'We need to create the manual now, where is this document?', to which the answer from the team is 'Eh... What?' […] The test cases also need to be based on something, for example."
- PdMb

Thus, the product manager indicated that in the cases where documents are used outside the development teams, the uses of those documents are not always clear to the developers. This view is somewhat substantiated by three developers (Da, Df, Dg), who were uncertain about the use of the PFD outside of development. For example, one developer (Da) said that nobody seems to know the use of the PFD or how it should be written. Three developers (Da, De, Df) expressed concerns with the PFD. Furthermore some developers (Da, Dd) thought the purpose of the PFD is unclear. However, as previously written, interviews with several testers (Tb, Tc) pointed to that the PFD is used in testing. Whether the developers' concerns are due to a lack of understanding or due to that the document itself is lacking in some way needs to be further evaluated.

Furthermore, one interview indicated that the process around the PFDs is sometimes not understood correctly. Specifically, one tester (Tc) thought it was not ideal that the PFDs are not maintained after the end of the project they were created in. However, according to other sources (including the company's process description as well as other interviews (Da, Db, Di), the PFDs are in fact maintained, as they are reference documentation. In this case, the tester (Tc) only used the PFDs at the project sites, which are not maintained after project closure, and not the ones located in Git. This could mean that outdated documentation sometimes is used when writing test cases due to confusion regarding the PFDs. However, as this was only an example from one person, it is hard to make any general conclusions. Therefore, it might be a good course of action to investigate the purpose and usage of the PFD, in order to

establish the intended way of writing and using it. This information could afterwards be conveyed to the employees at the company.

Lastly, a senior tester (Ta) suggested that not everyone thrives in the company's culture, which emphasizes face-to-face communication and the existence of unclear requirements emerging from the company's requirements process. The following fragments from an interview with the senior tester show some of these soft issues:

> *"Some people think it's uncomfortable to talk a lot with others, they would have felt much better to just get instructions of exactly what they should do. But I believe a lot in this agile principle, that you should have a living communication. So the tester should go talk to the orderer and ask what [he/she] really wants and talk to the developers why it is like that. Then the test gets created through a communication of both written requirements, prototypes from the developers and dialogue with the developers. […]*
>
> *Many on Axis think that they work according to a waterfall model, but in reality it is very iterative. Many think it is very unpleasant – they get mad at the orderer for changing [his/her] opinion and so on. So, the real problem is really people's attitude. If people had an attitude that agile is good, then suddenly we work a lot more agile than people think. […]*
>
> *"It's funny, because we have the culture on Axis. They talk a lot about the culture here, that you should be open and talk […]. The company encourages communication a lot. A lot of activities are organized to try to create that culture. But when we look at how we work in development, I think that we sometimes forget that and would rather just talk to each other via documents and text."*
> - Ta

These statements indicate that even though the culture may facilitate the company's requirements process, there are still some cultural challenges. Therefore, the company could reap benefits through further establishing the company's culture among its employees. As the processes and the culture affects each other directly, taking cultural aspects into account when changing the processes at the company is important.

6 Discussion

This section consists of three main parts. Firstly, the research questions for this study are elaborated on. Secondly, the limitations of the study are explained. Lastly,

the section includes a discussion about future work that can be performed in the light of this thesis.

6.1 Research questions

The section summarizes the main findings, condensed into the six research questions initially formulated. The research questions are answered both through these main findings, but also with a more general discussion.

6.1.1 RQ1: What constitutes the requirements process used at Axis Communications AB?

Through the detailed descriptions in section 0 "An elaboration on the company's requirements process", this research question has been answered implicitly already. However, here follows a concise description of the most characterizing elements of the company's requirements processs. The elements are summarized in Figure 3.

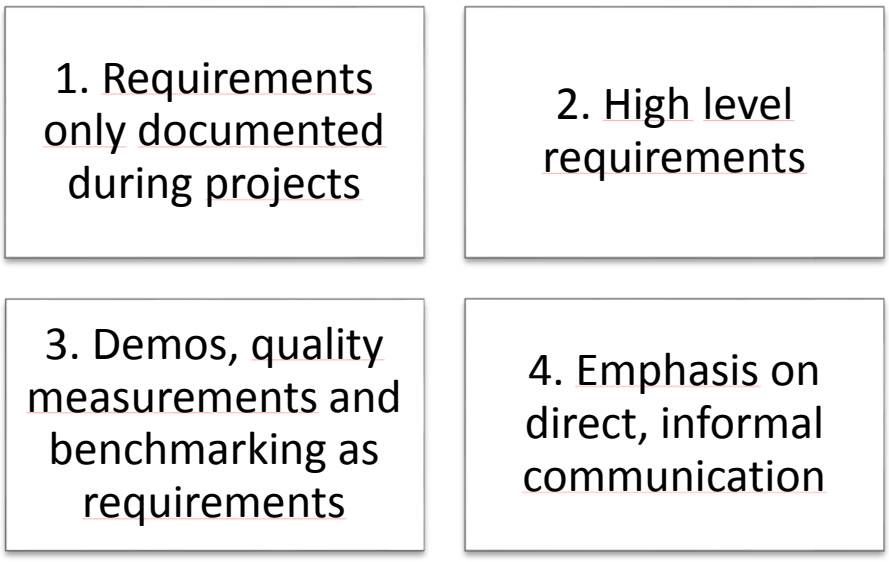

Figure 3. The main characteristics of the company's requirements process.

Firstly, the requirements are not always documented in a rigorous manner. Since no requirements documentation is maintained after projects are closed, the company does not have a comprehensive set of requirements, such as a requirements database, that reflects the current software. Requirements knowledge is instead acquired through other sources of information (see section 5.2), such as other documents, tests

and colleagues who may have more experience in the matter at hand. From an RE perspective, the requirements related documentation was found to have two primary purposes at the company (see section 5.9.2). The first purpose is to act as "handshake", a way of confirming that there is common understanding between the different stakeholders of a project. The second purpose is to act as reference documentation, storing important knowledge for future reference. However, as the requirements are not comprehensive and maintained, there is little requirements documentation acting as reference documentation. Instead, the closest alternative is the functional descriptions, which are maintained continuously.

Secondly, since the orderer in many cases specifies the project's task on a high level, explaining the problem that needs to be solved rather than how to solve it, the team can design its solution in the way that they see fit (see section 5.1). This means that the orderer contributes with higher level requirements input, whereas the team itself elicits the more detailed requirements.

Thirdly, the company uses a certain set of methods for dealing with requirements. These methods include the use of demos in projects as a way of getting quick feedback from the orderer, as well as the process for handling quality requirements. Since putting quality requirements on the platform is a complex activity, the company has started to measure quality aspects in the different products that are using the platform (see section 5.3.1). This allows the company to monitor the evolution of the quality aspects and catch trends of declining quality at an early phase. Additionally, the company uses benchmarking as requirements in some cases (see section 5.4). This means that earlier platform versions, or software in specific products, are used for comparison, allowing an orderer to specify new functionality in terms of old software. Although this study cannot conclude whether or not this approach can be seen as beneficial, it is an interesting approach. The reason for this is that the approach potentially could allow much requirements documentation to remain unwritten without significant drawbacks – saving time and money at the same time as facilitating further use of a process with limited requirements documentation.

Fourthly, the culture at the company focuses on openness as well as informal and direct communication, where face-to-face communication in particular is emphasized. In practice this is seen through an open climate at the company, where asking questions is always allowed, and a tight interaction between employees in general. A project team, for example, normally has close contact with its orderer, discussing the scope and the details of the project iteratively. This helps the team to overcome issues related to the order, which in many cases is written in a way that is perceived as unclear and vague by developers (see section 365.1).

6.1.2 *RQ2: Are there any factors at Axis Communications AB that facilitate software development with the current requirements process?*

In order to enable the use of the current requirements process, several important factors have been identified at the company. These factors relate primarily to cultural and organizational aspects and are summarized in Figure 4.

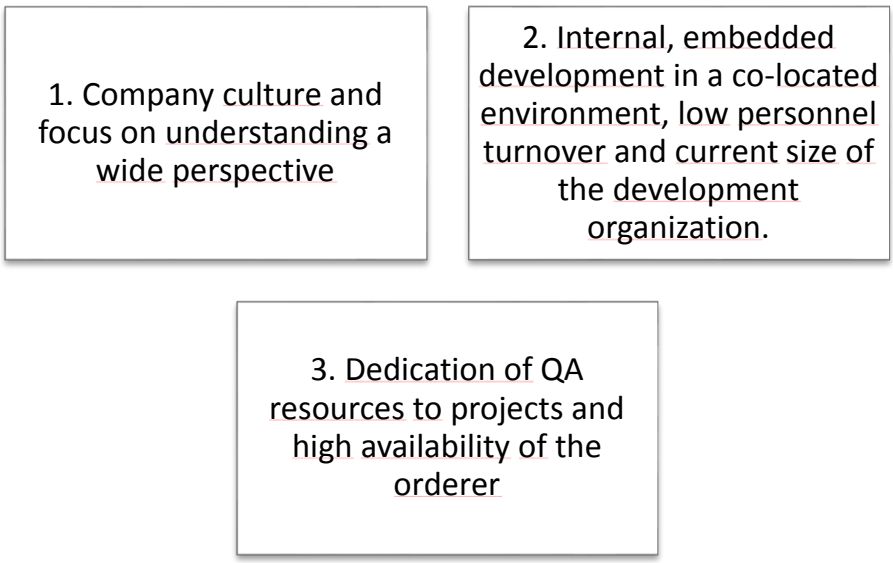

Figure 4. Summary of the facilitating factors found at the company.

Within the cultural area, the main factors include the nature of the communication between employees at the company (see section 5.9) and the emphasis on engineers to understand a wide perspective of the company's activities (see section 5.10.3). The communication inside the company is shaped by the culture of helpfulness and cooperation, which are traits included in one of the company's core values – Act as One. This creates a communication climate where, as previously mentioned, a lot of communication is done informally and face-to-face. The results also showed that many employees favor face-to-face communication over documentation and being communicative is seen as an important trait for engineers in the company (see section 5.9.1). As a whole, the communication seems to facilitate networking and knowledge sharing, which might be one of the reasons why the aspect of acquiring requirements information directly through colleagues, rather than through documentation, works well at the company. The concept of understanding the wide perspective includes aspects such as an understanding of the company's business model and other departments (see section 5.10.3). The emphasis on engineers to understand this perspective is reflected in the company's career ladder for engineers, a ladder that besides technical compe-

tency also focuses on this understanding. For more details, e.g. regarding what consequences this understanding has on the requirements process, see section 5.10.3.

Multiple additional factors, relating to the organizational dimension, have been identified in this study. On the highest level, the characteristics of the development play an important part. Specifically, the employees are doing internal, embedded development in a co-located environment, naturally giving some immediate consequences. For example, face-to-face communication is facilitated by co-located developers. Also, since the development is internal, less emphasis can be put on the requirements as a form of contractual agreement, compared to situations where development is conducted based on such a contract. Furthermore, the fact that the company has a low personnel turnover means less knowledge is lost due to employees who quit, reducing the dependence on documentation to mitigate such losses. It might also be possible that the scale of the development at the company influences the viability of the requirements process. For example, a more large scale software development company would perhaps need a more structured process to aid the coordination of its employees.

On a somewhat lower level, some factors are related to the organization of the departments and teams at the company. Specifically, the dedication of QA resources to projects means that more knowledge can be shared verbally (see section 5.10.2), reducing the need for documentation during the course of a project. Additionally, the availability of the orderer is in most of the cases high, which gives the team the opportunity of continuously clarifying what is demanded from them.

6.1.3 RQ3: What benefits does Axis Communications AB gain from using the current requirements process?

In this section, the benefits of the current requirements process are discussed. The benefits are summarized in Figure 5.

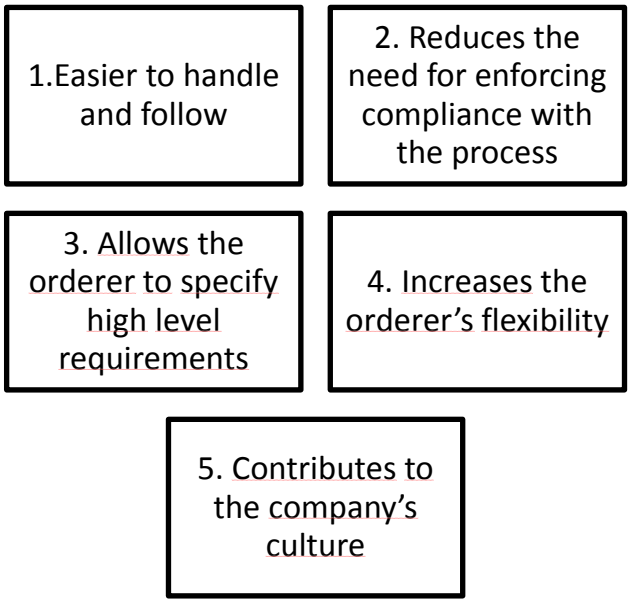

Figure 5. Summary of the benefits of the company's requirements process.

Some of the main benefits with having a process, in any scenario, with limited requirements documentation relate to that the process as such is easier to handle. Firstly, such a process is more clear and simple for the employees to understand and less complex for senior management to change, as there are fewer details to take into consideration. Furthermore, the company's requirements process is designed in such a way that the individual project teams get a certain amount of freedom to extend it according to their needs, e.g. a team can write more documentation than demanded by the process if the need arises. Therefore, the restrictions for each team are fewer with a process which enables each team to choose the approach that they feel is more efficient. This relates closely to the concept of self-organizing teams, which has been argued (see section 2.1 "Agile software development") to be an important factor for the success of agile projects. In this sense, the self-organization of teams at the company creates a potential for decentralized decision making and cost efficient work, while at the same time maintaining agility and enabling each team to alter its own extensions of the process.

Additionally, in a process with more detail and rigor there might be a need for ensuring that the work of each team is conducted according to the process. This can be done through information and education, but also through enforcement in different ways. Not having a process with many details and rigor means fewer resources have to be put toward this purpose. Moreover, as there is no single best process that fits in

all teams simultaneously and takes the different conditions of all teams into account, some variance in how the process is used in practice might be necessary. Through allowing different teams to work in somewhat different ways, this challenge is addressed in a way that is significantly less resource intensive.

However, the company's requirements process also has other, more apparent, benefits. First of all, the orderers are allowed to specify high level requirements and leave the details of how to implement them to the project team (see section 5.1). Through this approach, the orderer can focus more on other areas, e.g. keeping track of the customers' needs. The team, being specialized in their specific area, is at the same time allowed to design the best possible solution from their perspective, which could improve the quality and/or reduce the cost of the solution.

Also, due to the high level of the order that is written initially in a project (see section 5.1), the orderer gains some flexibility in changing the scope or the requirements of the task during the project. This flexibility is most apparent in comparison with more rigorous requirements documentation methods, where changing requirements become more cumbersome to do as the requirements are already specified in detail. According to previous research, see section 2.3 ("Software Product Line Engineering"), this type of flexibility is beneficial particularly in environments with market volatility, where requirements change suddenly and/or frequently.

Besides these benefits, relating to the division of work between a project and the orderer, there are also some benefits which relate more to the culture of the company. The company's requirements process can be argued to contribute to the current company culture, which just as the requirements process focuses on face-to-face communication, helpfulness and interaction, rather than communication through documentation. It was indicated that using direct communication gives positive side-effects such as networking between employees and an increased understanding of other parts in the organization. Additionally, the requirements process facilitates knowledge sharing, both between teams and between departments (see section 5.10.2).

6.1.4 RQ4: What challenges does Axis Communications AB face due to the use of the current requirements process?

In the data, many challenges were found. The most relevant ones are brought up here and discussed. A summary of the most significant challenges can be found in Figure 6. Interestingly, the challenges are in more detail than the benefits. The reason for this might be that interviewees in general find it easier to perceive challenges than to perceive what benefits they receive through the process.

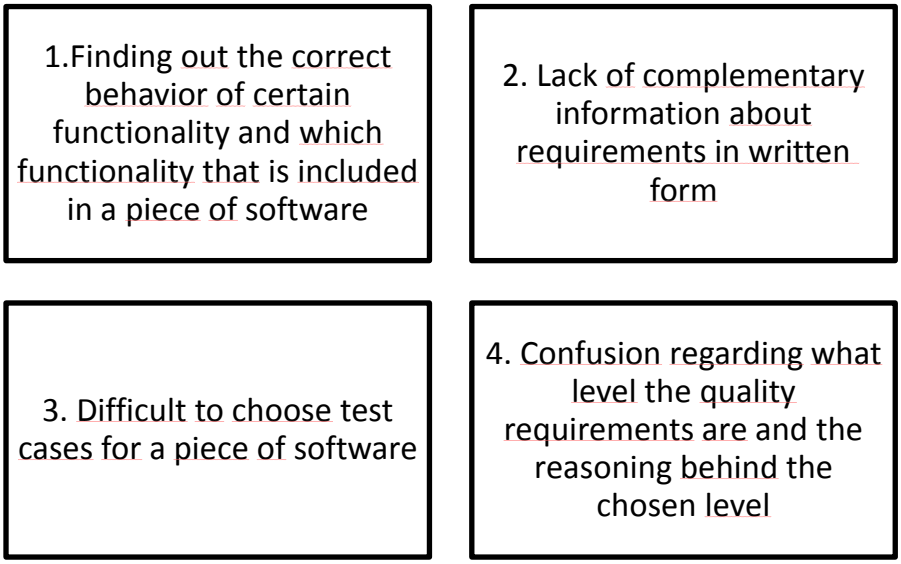

Figure 6. Summary of the main challenges found in the study.

Corresponding to the section 5.5 "Implications of the company's requirements process", several issues were found relating directly to the company's requirements process. Firstly, finding out both what the correct behavior of certain functionality is (see section 5.5.1), as well as which of the platform's functionality that is included in a piece of software (see section 5.5.2), seemed to create some difficulty for the employees. However, the latter was not substantiated by any developers, which constituted the majority of the interviewees. Hence, the interview data for this difficulty is not comprehensive, even though the difficulty was expressed by representatives from both QA and Product Management.

Secondly, complementary information about requirements was desired by several interviewees (see section 5.5.3). This type of information, such as who ordered the functionality, why it is important and who it is important for, is in many cases not available in written form. Thus, employees have to depend on individuals in order to acquire this knowledge, in some cases causing difficulties in acquiring the information. According to the interview data, this could create issues when trying to remove unwanted functionality, as the purpose and usage areas of the functionality is hard to understand without that kind of information.

Thirdly, interviews with QA representatives indicated that there are difficulties in choosing test cases for a piece of software. This hardship is related to the absence of traceability between different software and tests, the lack of historical test data and the aforementioned difficulty in knowing what functionality is included in a piece of

software (for details see section 5.5.4 “Choosing what test cases to run”). One of the reasons for this challenge is that there is a certain complexity in choosing test cases for a piece of software, as the actual functionality in the software varies with several factors, such as what product and platform version is being tested. Although a more rigorous documentation of requirements, including traceability to products, platform versions and test cases, could facilitate the process of choosing test cases, maintaining that kind of traceability is presumably neither an easy nor a cheap task. As the company’s own ongoing improvements, specifically the development of the feature lists, have the potential to mitigate this challenge, it might be reasonable to reevaluate the challenge after the implementations of these improvements are complete.

Another challenging area in general is the quality requirements at the company (see section 5.3). This study has shown that there is confusion regarding what the level of the quality attributes as well as what the reason for selecting the specific values are. One explanation for this might be that putting quality requirements on the platform is a complex activity, e.g. due to the variability of the products (different functionality and different hardware in many combinations). Additionally, the demands from the customers increase steadily over time, causing quality aspects like performance to need constant improvement. This contributes to making the specification of quality requirements a difficult and perhaps cumbersome activity. In turn, this might be a reason for the fact that the company is attempting to manage quality aspects through measuring them directly on the product, thus getting an overview of how the software performs in the different settings. However, as this approach was questioned by a few of the interviewees from QA, it is not certain that the approach is appropriate. Future work is needed in this regard to verify whether or not measurements are an adequate method for managing evolution of quality aspects.

Finally, two challenges relating to soft factors within the company were explored in section 5.10. Firstly, according to the literature found in the literature review, the introduction of new employees was presented as one factor which might be troublesome in a requirements setting with limited documentation (see section 2.2.3 “The challenges of Agile Requirements Engineering”). However, the most newly employed interviewees at the company did not consider the absence of requirements to be one of the main challenges for new employees (see section 5.10.1). Rather, the challenges for new employees seemed to mainly relate to other topics, such as tools and integration procedures, which cannot be facilitated by requirements documentation. Thus, although some interviewees indicated that newly employed require more time than necessary from experienced employees, it cannot be concluded whether or not more rigorous requirements documentation would make the introduction of new employees more efficient.

Secondly, several facts point to that the dependence on individuals at the company is high, due to the low amount of requirements documentation. For example, the high

level of the order documents cause dependence on the orderers, as the teams need further information in order to conduct the projects in an appropriate manner. Also, the difficulties in finding the correct behavior through documentation can be argued to lead to a dependence on the individuals who can explain what the correct behavior is. While some interviewees pointed to this dependence as a risk for the company, likely causing knowledge loss when employees quit or become available, other interview data suggested that the challenge is mitigated in a number of ways (see section 5.10.2). As this topic was explored, it was found that some employees did not perceive this as a major risk. The reason for this might be that the dependence is mitigated through factors such as knowledge sharing and low personnel turnover. Therefore, even if knowledgeable employees are always a major asset for companies, it is hard to assess the significance of this challenge and whether or not more rigorous requirements documentation would provide any significant increase in the mitigation of the knowledge loss.

6.1.5 RQ5: What can be said about the scalability of the requirements process used at Axis Communications AB?

This topic is mainly treated in the section 5.7 "The scalability of the requirements process", where the analysis of the collected data has been done in more detail. From this analysis, it is clear that employees do not see scalability as a major issue at the company, at least not due to the company's requirements process. Instead, a limited amount of documentation was seen as a condition for keeping the process scalable. Therefore, scalability due to company growth does not seem to be a major risk for the company.

However, one can argue that it will be hard to continue using the company's requirements process if more employees conduct distributed development. Distributed development might make it difficult to use direct communication as means of sharing requirements knowledge. In this case, more rigorous requirements documentation might be warranted, as it could be used to substitute the current approach of emphasizing direct communication. In other words, requirements documentation could be written as a way of reducing the dependence on direct communication, which might be harder to have in a distributed setting.

Still, it seems like the company is aware of the general challenges associated with scalability and is taking measures in order to mitigate them. For example, the company uses architectural refactoring, thus reducing the dependence between different teams and thereby also the amount of questions (e.g. about requirements) that needs to be asked between them. Also, the company has implemented several modifications both to the development process and to the organization in recent years due to its rapid growth. All in all, this points to that the company is in control of scalability and the issues that might be associated to it.

6.1.6 *RQ6: Would the implementation of a requirements database be a viable option for Axis Communications AB?*

Implementing a requirements database is perhaps the most obvious undertaking from an RE point of view. From this perspective, a requirements database would improve several aspects of how the requirements are handled. However, whether a requirements database would actually be beneficial for the company from the larger business perspective is not certain. There are also organizational challenges relating to implementation. Here follows some background on the topic as well as the implications that an implementation of a requirements database would have, including the work needed to specify the requirements, the costs of implementing a tool and the organizational challenges.

In the current process, no requirements documentation is maintained after a project is completed. From a requirements perspective, the most interesting artifacts that are currently being maintained past a project's closure are test cases, functional descriptions and the products themselves. However, none of these artifacts can tell whether or not the current behavior of the software is the intended one. Some interviewees claimed that test cases are doing this (see the last quote in section 5.2), but the challenge in this regard is to handle a situation where confusion arises around the test cases' correctness. The interviews have shown that a common way for clearing confusion is simply asking a relevant person for guidance. Though many at the company are content with this approach, it creates a certain dependence on individuals (see section 5.10.2).

One way to address these challenges could be through having more clear definitions of the requirements and storing them in a requirements database. However, investigating the detailed implications of implementing a requirements database is a complex task, simply due to the many parties that potentially will interact with it and the complexity of estimating how their work will be affected. The complexity is further increased due to the complicated nature of the organization, the platform and the different products that the platform adapts to. Handling the variability of the platform, depending on which product it is implemented in, is not trivial from an RE perspective.

The implementation of a requirements database would also mean a challenging organizational change for the company. The costs of its implementation could potentially become high. This is especially due to the low amount of clearly specified requirements in the current software, which would create the need to "elicit" the requirements on the current platform again in order to specify them. As older versions of the platform are in use by customers, there is some motivation for specifying the requirements also for the older versions. Unfortunately, doing this for any older platform versions would increase the costs even further. Additionally, there are also costs tied to implementation of a requirements management tool, which likely would be

needed in order to handle the sizable amount of requirements that the company currently has. For example, these costs include researching and choosing a tool, implementing the tool with regards to both infrastructure and training of employees, as well as possible costs associated to the tool directly (such as license fees).

However, it is not certain that even a successful implementation of a requirements database would be beneficial for the company. The interviews have shown that the culture at the company tends to emphasize face-to-face communication. A requirements database would likely not facilitate this emphasis, as it would probably mean a significant increase in written requirements. Respectively, since a requirements database emphasizes this written form of requirements rather than face-to-face communication, a database might not be facilitated by the company's culture to the same extent that the current process is.

Also, even though a requirements database could improve certain activities relating to requirements, for example making the creation of test cases more efficient, it would at the same time mean a significant amount of extra work. This work would include implementation of the infrastructure, extracting all the implicit requirements currently present and then maintaining them. Furthermore, due to the culture at the company and the general focus on keeping the processes light, it seems there are some challenges to overcome relating to human factors. For example, the impression that the authors of this report have gotten during the interviews is that there is a general resistance against heavy processes and documentation. This indicates that many in the company would not welcome a requirements database, increasing the difficulty of successfully implementing one.

Due to these various difficulties of implementing and maintaining a requirements database, the choice of whether to add a requirements database or not is not trivial. Any requirements database that is implemented should still be light, in order to reduce the difficulties. However, challenges still remain, such as the division of responsibilities relating to a requirements database and changing organizational routines and practices around it. As these difficulties are both uncertain and possibly large, the viability of implementing even a light requirements database should be further investigated.

Another possible approach would be to do smaller modifications to the documentation in order to improve the general requirements process. One of these modifications could be to collect requirements related documentation such as the PFDs and the CDs in a common place at the company's intranet. This could be a viable option, as the cost for implementing this would likely be low. As of now, PFDs are stored both on the intranet and in Git, but it is mainly the PFDs in Git that are updated. Since the interviews gave an example of that some people outside the platform development department use the PFDs at the intranet it can therefore be a good idea to collect the PFDs in a common place where they are kept updated. This would also make it easier

to get an overview of the available functionality, somewhat mitigating the issues with knowing what functionality is included in the LFP. As developers are not as concerned with getting an overview, a centralized place where the documents are stored would primarily help people outside of the development organization.

6.2 Threats to validity and limitations

This section addresses the different threats to validity and limitations that were present in the study. As the interviews were captured in transcriptions, the main threat to *description validity* is that the transcriptions did not reflect the interviews. To deal with this threat, each interview was recorded and the transcriptions were thereafter based on those recordings. Reliability of the transcriptions was further enhanced through sending the interview transcriptions back to the interviewees, who could then confirm their correctness. Additionally, the interviews were anonymous in order to encourage honesty among the interviewees.

As for *interpretation validity*, the main threats were the formulation of questions during the interviews as well as the analysis of the transcriptions. In order to handle these threats, care was taken to avoid conscious and unconscious *observer bias*. For this purpose, special effort was put towards assessing the openness of the questions which were asked early during the interviews, reducing the risk of imposing the researchers' theories on the interviewees. Discussions were also held continuously during the study to challenge the assumptions of each researcher. Additionally, the interview instrument was reviewed by several outside parties in order to ensure its appropriateness. Regarding the data analysis, specifically the formulation of statements and assertions explained in section 4.2.3, the main threat to validity consisted of wrongly interpreting the transcriptions. However, this was dealt with through the concept of constant comparison, confirming statements and assertions continuously by comparing them with the interview transcriptions. Finally, special care was also taken to avoid the risk of presenting any quotes outside of their context. In practice, this was done through revisiting the recordings each time a quote was extracted. Any uncertainties in this regard was brought up explicitly by the researchers and addressed in cooperation.

Internal generalizability was addressed through including interviewees with different experience and roles, from several different departments, in the data collection phase. Nonetheless, not all perspectives might have been explored in greater detail from an organizational point of view, as this was unfeasible due to the size and complexity of the company. Considering *external generalizability*, the main restriction is the fact that only one company was included in the study. Thus, the results in this thesis have not been tested in other contexts and are therefore not generalizable for all companies conducting software development. However, creating a comprehensive theory that could be used by the software industry or research community was not the aim of this thesis.

Besides the above threats to validity, three general limitations were also identified in the study. Firstly, since the scope of the research questions has been wide, more efforts have been put towards getting a rich set of data on the research questions. This was done through qualitative data collection methods. Because of the width of the research questions, detailed exploration on some aspects had to be left out. Another implication of this wide focus was that the concept of theoretical sampling in Grounded Theory was used. Adopting theoretical sampling in this study meant constantly striving for more details and deeper understanding, rather than to repeat the same questions in order to quantify the concepts already identified. Because of the use of this concept, this study followed a principle where more than one occurrence of a statement was considered to fulfill saturation. The purpose of presenting the quantifications in this report is mainly to highlight if it was one or more interviewees who expressed a certain thing. In cases where there were several interviewees these were presented with their corresponding code in order to convey role and approximate experience. Because of the above, the reader of this thesis needs to be advised that any interpretation that is based on the quantifications of the interview results in this study must be performed with great caution. It should, for example, be made clear that the quantifications are not statistically significant, as verifying the significance of the identified concepts was not within the scope of this exploratory study. Whether or not this saturation strategy is a feasible approach in this context remains to be shown by future studies.

Secondly, certain aspects of Grounded Theory could not be adopted. One of these aspects is the concept of theoretical saturation [16], achieved through continued data collection until additional data does not add any value to the theoretical model being developed. Given the scope of this thesis, theoretical saturation would not have been achieved without conducting a significant amount of additional interviews. Furthermore, Grounded Theory focuses on social aspects, such as values, beliefs and behavior, and not on processes, costs and efficiency, which were the main topics of this thesis. Finally, this study did not aim to build a theoretical model, which is the fundamental purpose of Grounded Theory. Based on these reasons, applying Grounded Theory as described by Corbin and Strauss [16] would not have been feasible in this context. Given the constraints, the methodology was modified during the study in order to adjust it to the width of the research questions. However, as the modifications were tailored to the specific context of this study, the methodology used in this report has not been validated by the research community.

Thirdly, the only process being evaluated in this report was the requirements process as used by the interviewees, rather than the written one. This means that no evaluation of the specified requirements process and how it compares to the actual process were done, which is commonly done when conducting case studies in the field. The reason for not doing a comparison between the specified process and the actual was

that the platform development organization lacked a clear definition of its requirements process, making a comparison impossible. Due to this constraint, this research did not elaborate on how well the specified requirements process in the platform development organization is followed by its employees.

6.3 Future work

As this study has a broad scope as well as an exploratory approach, not all aspects that are presented have been thoroughly verified. Thus, future work is needed to verify these aspects. In the report where these aspects are presented, the need for additional verification has in many cases been brought up explicitly.

In order to further validate the findings of this study, more data could be collected from other parts of the organization. For example, conducting interviews in departments such as NVP, System Applications and Technical Information Management would give a more comprehensive view of how the requirements process affects the different parts of the company.

Furthermore, assessing the viability of using benchmarking as requirements, as this company is doing it, provides an intriguing area for further research. Through additional work, the benefits and the drawbacks can be revealed. Also, the interesting question of whether or not such an approach can aid the minimization of costs, both short term and long term, in a development organization can be answered.

Finally, future work is needed to assess the applicability of the company's requirements process in other contexts. This would make it possible to conclude whether or not all parts of the unique context that Axis Communications AB has, is needed for the use of the process. It would also be interesting to investigate the similarities between the way this company is working and that of the development in open source communities [53], in order to assess the applicability of open source development practices in proprietary development.

7 Conclusion

This thesis was conducted in order to explore and evaluate the RE process used at Axis Communications AB. Thus, the study has described the requirements process used at Axis and the factors that facilitate it. Also, the main benefits and challenges that are linked to this process were presented. Lastly, the scalability of the process as well as the viability of a requirements database in the company's context were elaborated on in this thesis.

The research has shown how the company handles requirements in a context where SPLE concepts are combined with agile software development. The company uses a

requirements process where no requirements documentation is maintained over time. Rather, other sources of information, such as the employees' individual knowledge, are used in order to convey the requirements of the software. Based on the exploration of previous research relating to lightweight requirements, the process seems to be lightweight. Furthermore, the process is facilitated by a number of different factors, including the company's culture and organization, and was shown to yield several benefits and challenges. Regardless, scalability does currently not seem to represent a major challenge and the company is taking measures in order to keep the process scalable. Regarding the viability of a requirements database in the company's context this thesis concludes that the implementation of a requirements database is not a trivial task. Instead, smaller modifications to the process could be an alternative option for the company.

The company's requirements process can be recommended in contexts where it is applicable, since it has the potential to reduce the costs associated to having a more rigorous requirements process. However, the contexts where this process is applicable could demand several of the characteristics that are present in the company. For example, internal, embedded development in a co-located environment with a company culture that facilitates communication could be important factors to have. At the very least, the authors of this report would suggest that such a requirements process is only applied in a setting where the development is internal and developers are co-located. Also, due to the difficulties shown in the testing area, the process should perhaps not be used in a setting where undiscovered defects can cause substantial damage or loss of life. One reason for this recommendation is that previous research has showed that more documentation, with greater detail, is needed in these contexts (see section 1 "Agile Requirements Engineering and its practices").

This thesis has presented several interesting concepts used in the company's requirements process. These include the use of benchmarking and products as requirements. The thesis has also elaborated on how quality requirements are managed in a company that develops both software and hardware in a context where SPLE is combined with agile software development. Finally, since the company has increased the amount of other kinds of documentation as a result of growing in size, additional requirements related documentation might be necessary at some point in time. Currently, however, it seems that the company intends on doing this through functional descriptions and testing artifacts, rather than through traditional requirements documentation.

8 References

[1] Adam, S.; Doerr, J. & Eisenbarth, M. (2009), *Lessons Learned from Best Practice-Oriented Process Improvement in Requirements Engineering: A Glance into Current Industrial RE Application*, in 'Requirements Engineering Education and Training (REET), 2009 Fourth International Workshop', pp. 1-5.

[2] Adam, S.; Riegel, N. & Gross, A.Richard Berntsson Svensson, Daniel Berry, (2012), *Focusing on the "Right" Requirements by Considering Information Needs, Priorities, and Constraints*, Chapter 3, Requirements Engineering Efficiency Workshop (REEW), pp. 68-74.

[3] Alford, W. & Lawson, J. (1979), *Software Requirements Engineering Methodology (development)*, Rome Air Development Center, Air Force Systems Command.

[4] Ali Babar, M.; Ihme, T. & Pikkarainen, M. (2009), *An Industrial Case of Exploiting Product Line Architectures in Agile Software Development*, in 'Proceedings of the 13th International Software Product Line Conference', Carnegie Mellon University, Pittsburgh, PA, USA, pp. 171-179.

[5] Ambler, S. (2012), *What does it mean to be "Big"? The agile scaling model*, *Cutter IT Journal* **25**(2), 6 - 10.

[6] Bakalova Z., Daneva M., Herrmann A. & Wieringa R. (2011), *Agile Requirements Prioritization: What Happens in Practice and What Is Described in Literature*, Requirements Engineering: Foundation for Software Quality, Springer, Section 7, pp. 181-195.

[7] Batool, A.; Motla, Y.; Hamid, B.; Asghar, S.; Riaz, M.; Mukhtar, M. & Ahmed, M. (2013), *Comparative study of traditional requirement engineering and Agile requirement engineering*, in 'Advanced Communication Technology (ICACT), 2013 15th International Conference', pp. 1006-1014.

[8] Beck, K. (1999), *Embracing change with extreme programming*, *Computer* **32**(10), 70 - 77.

[9] Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J. & Thomas, D. (2001), *Manifesto for Agile Software Development*, URL: http://agilemanifesto.org/

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

[11] Blau, B. & Hildenbrand, T. (2011), *Product Line Engineering in Large-Scale Lean and Agile Software Product Development Environments - Towards a Hybrid Approach to Decentral Control and Managed Reuse*, in '2011 Sixth International Conference on Availability, Reliability and Security', pp. 404 - 8.

[12] Boehm, B. (2000), *Requirements that handle IKIWISI, COTS, and rapid change*, Computer **33**(7), 99 - 102.

[13] Bose, S.; Kurhekar, M. & Ghoshal, J. (2008), *Agile methodology in requirements engineering*.

[14] Cao, L. & Ramesh, B. (2008), *Agile requirements engineering practices: An empirical study*, IEEE Software **25**(1), 60 - 67.

[15] Cockburn, A. (2000), *Selecting a project's methodology*, IEEE Software **17**(4), 64 - 71.

[16] Corbin, J. & Strauss, A. (2008), *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, SAGE Publications.

[17] De Lucia, A. & Qusef, A. (2010), *Requirements Engineering in Agile Software Development*, Journal of Emerging Technologies in Web Intelligence **2**(3), 212 - 20.

[18] Delgadillo, L. & Gotel, O. (2007), *Story-wall: a concept for lightweight requirements management*, in '2007 IEEE International Conference on Requirements Engineering', pp. 377 - 8.
[19] Diaz, J.; Perez, J.; Alarcon, P. & Garbajosa, J. (2011), *Agile product line engineering: a systematic literature review*, Software: Practice and Experience **41**(8), 921 - 41.
[20] Diaz, J.; Perez, J.; Yague, A. & Garbajosa, J. (2011), *Tailoring the scrum development process to address agile product line engineering*, in 'Actas de las 16th Jornadas de Ingenieria del Software y Bases de Datos, JISBD 2011', pp. 457 - 470.
[21] Doerr, J.; Kerkow, D.; Koenig, T.; Olsson, T. & Suzuki, T. (2005), *Non-functional requirements in industry - Three case studies adopting an experience-based NFR method*, in 'Proceedings of the IEEE International Conference on Requirements Engineering', pp. 373 - 382.
[22] Eberlein, A. & Leite, J. (2002), *Agile requirements definition: A view from requirements engineering*.
[23] Ernst, N. & Murphy, G. (2012), *Case studies in just-in-time requirements analysis*, in 'Proceedings of the 2012 IEEE Second International Workshop on Empirical Requirements Engineering (EmpiRE)', pp. 25 - 32.
[24] Farid, W. (2012), *The NORMAP Methodology: Lightweight Engineering of Non-functional Requirements for Agile Processes*, in 'Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference (APSEC)', pp. 322 - 5.
[25] Farid, W. & Mitropoulos, F. (2012), *Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes*, in '2012 Proceedings of IEEE Southeastcon'.
[26] Forward, A. & Lethbridge, T. C. (2002), *The Relevance of Software Documentation, Tools and Technologies: A Survey*, in 'Proceedings of the 2002 ACM Symposium on Document Engineering', pp. 26 - 33.
[27] Gallardo-Valencia, R. & Sim, S. (2009), *Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile*, in '2009 Second International Workshop on Managing Requirements Knowledge (MARK 2009)'.
[28] Gorschek, T. & Wohlin, C. (2005), *Requirements Abstraction Model*, Requirements Engineering **11**(1), 79-101.
[29] Heikkila, V.; Rautiainen, K. & Jansen, S. (2010), *A revelatory case study on scaling agile release planning*, in 'Proceedings - 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010', pp. 289 - 296.
[30] Hoda, R.; Noble, J. & Marshall, S. (2012), *Documentation strategies on agile software development projects*, International Journal of Agile and Extreme Software Development **1**(1), 23 - 37.
[31] Hoda, R.; Noble, J. & Marshall, S. (2013), *Self-organizing roles on agile software development teams*, IEEE Transactions on Software Engineering **39**(3), 422 - 444.
[32] Hoda, R.; Noble, J. & Marshall, S. (2011), *Supporting self-organizing agile teams what's senior management got to do with it?*, in 'Lecture Notes in Business Information Processing', pp. 73 - 87.
[33] Hoda, R.; Noble, J. & Marshall, S. (2011), *How much is just enough? Some documentation patterns on Agile projects*, in 'ACM International Conference Proceeding Series'.
[34] Hofmann, H. F. & Lehner, F. (2001), *Requirements Engineering as a Success Factor in Software Projects.*, IEEE Software **18**(4), 58-66.
[35] Jacobs E. (2012), *"Executive Brief: Tracking Trends in Employee Turnover",* Society for Human Resource Management.
[36] Kovitz, B. (2003), *Hidden skills that support phased and agile requirements engineering*, Requirements Engineering **8**(2), 135 - 41.
[37] Lauesen, S. (2001), *Software Requirements: Styles and Techniques*, Pearson Education.

[38] Maiden, N. & Jones, S. (2010), *Agile Requirements - Can We Have Our Cake and Eat It Too?*, IEEE Software **27**(3), 87 - 8.
[39] Middleton, P. (2001), *Lean software development: two case studies*, Software Quality Journal **9**(4), 241 - 52.
[40] Mohan, K.; Ramesh, B. & Sugumaran, V. (2010), *Integrating software product line engineering and agile development*, IEEE Software **27**(3), 48 - 55.
[41] Nanthaamornphong, A.; Morris, K.; Rouson, D. W. I. & Michelsen, H. A. (2013), *A case study: Agile development in the community laser-induced incandescence modeling environment (CLiiME)*, in '2013 5th International Workshop on Software Engineering for Computational Science and Engineering, SE-CSE 2013 - Proceedings', pp. 9 - 18.
[42] Nawrocki, J.; Jasinski, M.; Walter, B. & Wojciechowski, A. (2002), *Extreme programming modified: embrace requirements engineering practices*, in 'Proceedings IEEE Joint International Conference on Requirements Engineering', pp. 303 - 10.
[43] Paetsch, F.; Eberlein, A. & Maurer, F. (2003), *Requirements engineering and agile software development*, in 'Proceedings of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises', pp. 308 - 13.
[44] Racheva, Z. & Daneva, M. (2010), *How Do Real Options Concepts Fit in Agile Requirements Engineering?*, in 'Proceedings 8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010)', pp. 231 - 8.
[45] Racheva, Z.; Daneva, M. & Buglione, L. (2008), *Supporting the dynamic reprioritization of requirements in agile development of software products*, in '2008 2nd International Workshop on Software Product Management, ISWPM'08'.
[46] Ramesh, B.; Cao, L. & Baskerville, R. (2010), *Agile requirements engineering practices and challenges: an empirical study*, Information Systems Journal **20**(5), 449 - 480.
[47] Rekaby, A. & Soliman, M. (2011), *Towards Intermediate-Agile Model Based On Agile Through Requirement Management And Development Enhancements*, in 'Proceedings of the 2011 International Conference on Software Engineering Research & Practice (SERP 2011)', pp. 686 - 90.
[48] Robson, C. (2002), *Real World Research - A Resource for Social Scientists and Practitioner-Researchers*, Blackwell Publishing, Malden.
[49] Rubin, E. & Rubin, H. (2011), *Supporting agile software development through active documentation*, *Requirements Engineering* **16**(2), 117 - 132.
[50] Runeson, P.; Host, M.; Rainer, A. & Regnell, B. (2012), *Case Study Research in Software Engineering: Guidelines and Examples*, Wiley Publishing.
[51] Salvaneschi, P. (2009), *Managing knowledge for information system evolution: the MinimalEDoc methodology*, Software Process: Improvement and Practice **14**(6), 337 - 47.
[52] Savolainen, J.; Kuusela, J. & Vilavaara, A. (2010), *Transition to agile development rediscovery of important requirements engineering practices*, in Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010, pp. 289 - 294.
[53] Scacchi, W., (2008), *Understanding Requirements for Open Source Software*, Design Requirements Engineering: A Ten-Year Perspective, Springer-Verlag, pp. 467-494.
[54] Sillitti, A.; Ceschi, M.; Russo, B. & Succi, G. (2005), *Managing uncertainty in requirements: A survey in documentation-driven and Agile companies*, in 'Proceedings - International Software Metrics Symposium', pp. 145 - 154.
[55] Sillitti, A. & Succi, G.Aurum, A. & Wohlin, C., ed., (2005), *Requirements engineering for agile methods*, Engineering and managing software requirements, Springer, chapter 14, pp. 309-326.
[56] Vanhanen, J.; Mantyla, M. & Itkonen, J. (2009), *Lightweight Elicitation and Analysis of Software Product Quality Goals A Multiple Industrial Case Study*, in '2009 Third International Workshop on Software Product Management (IWSPM 2009)', pp. 42 - 52.

[57] Vlaanderen, K.; Brinkkemper, S.; Jansen, S. & Jaspers, E. (2009), *The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management*, in '2009 Third International Workshop on Software Product Management (IWSPM 2009)', pp. 1 - 10.

[58] Waldmann, B. (2011), *There's never enough time: Doing requirements under resource constraints, and what requirements engineering can learn from agile development*, *in 'Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011'*, pp. 301 - 305.

[59] Wiegers, K. E. (2003), *Software Requirements*, Microsoft Press, Redmond, WA, USA.

[60] Zhang, Z.; Arvela, M.; Berki, E.; Muhonen, M.; Nummenmaa, J. & Poranen, T. (Sept. 2010), *Towards Lightweight Requirements Documentation*, Journal of Software Engineering and Applications **3**(9), 882 - 9.

A. Division of responsibility during the thesis work

The work in this master's thesis was done in a collaborative way, where both individuals participated in all activities. The activities as such were for the most part conducted with much discussion and with close to equal share of the work effort. However, in order to give an overview of which person was more involved in the different activities, this section presents the different activities together with the name of the person with the main responsibility. The division is found in Table 4 below.

Table 4. Division of responsibilities between the authors of the study.

Activity	Name of main responsible
Pre-study of company	Linus Ahlberg
Literature review	Johannes Persson
Interview instrument design	Linus Ahlberg
Interview conducting	Johannes Persson
Interview transcription	Linus Ahlberg
Interview analysis	Johannes Persson
Introduction (report section)	Linus Ahlberg
Background (report section)	Linus Ahlberg
Case company (report section)	Johannes Persson
Methodology (report section)	Linus Ahlberg
An elaboration on the company's requirements process, section 5.1-5.4 (report section)	Johannes Persson
An elaboration on the company's requirements process, section 5.5-5.10 (report section)	Linus Ahlberg
Discussion (report section)	Johannes Persson
Conclusion (report section)	Linus Ahlberg
Appendices, Bibliography, Layout	Linus Ahlberg
First page, abstract and miscellaneous	Johannes Persson
Presentation preparations	Johannes Persson

B. Interview instrument

This section contains the three interview guides used as a basis for the interviews that were conducted in the study. As the interviews were semi-structured, not all questions asked are specified here and not all questions that are written here may have been asked during all interviews.

B.1 For developers

Personal
1. What are your main duties and responsibilities? 2. What is your experience? For how long have you been working in your current role?
The requirements process
3. Draw the requirements process on the board based on how you perceive it. How do you get requirements? What do you do with the requirements? Do you forward the requirements to anyone? Which of the requirements are documented? 4. Do you experience any problems due to the process of documenting requirements?
Interface towards orderers
5. How do your projects communicate with the orderer? 6. On what level are the requirements that your projects get from the orderer? Is some information missing? Is it a problem? 7. How do your projects refine high level requirements? Are the refined requirements stored anywhere and are they communicated back to the orderer? 8. How do your projects validate their output with the orderer? 9. Do your projects use prototypes? Does it work well?
Documentation
10. Which documents do you have in your projects? 11. Which documents must your projects produce? 12. Which documents do you use and how often do you use them? 13. What is the purpose of each document? Are all the documents and parts of the documents necessary? 14. Do the documents contain any duplicated information? 15. Is the quality of the documents good enough for you to be able to work efficiently? 16. Are you missing any documentation that could have been useful to you?
Knowledge sharing
17. What do you do when you need to know the functionality of the software? Would more documentation be useful in this regard? 18. Do you have to answer many questions about how the software is working? Who asks? Is it a problem? 19. Are there any typical questions? Could these questions be answered through

documentation? 20. Would you save time through documenting the answers rather than to answer them yourself? Would the person asking the question save time? Overall, would the organization benefit from documenting more? 21. Would it help you if other employees documented more? Would you rather go through documentation than ask questions? Overall, would the organization benefit from documenting more? 22. Is it possible to find out why a feature behaves the way it does? If not, is this a problem? Do you need to know this? 23. Does the amount of questions increase as the company grows? 24. Do you see any risks, with regards to scalability, to keep working the way the company does with requirements documentation? 25. How do your projects communicate with QA? How do they know the correct behavior of the software when testing it? 26. How does your department introduce new employees? Do they experience any problems due to the amount of requirements documentation that is available? 27. How is knowledge about changes in the software's behavior communicated to stakeholders? 28. Do you think any knowledge that is shared verbally needs to be documented? 29. Do you think it is a risk that much knowledge about the software platform is individual and not documented anywhere?
Other
30. Do your projects have any process that addresses quality requirements? If not, how are these handled?
Extra
31. Do you experience that others complain about the documentation? 32. Is project specific documentation maintained? Why/why not? Do you need to maintain documentation to the extent that you do? 33. Do you experience any other problems relating to the amount of requirements documentation? 34. Do other teams in your department work differently than your team does? 35. Do you see any connection between the amount of documentation in a project and the successfulness of the project? 36. How many functional areas are usually affected by a typical project? 37. How is the platform organization affected by the product organization?

B.2 For testers

Personal
1. What are your main duties and responsibilities? 2. What is your experience? For how long have you been working in your current role?

The company process
3. Based on what (e.g. documentation/communication) do you create your test cases? 4. Do tests fail because QA have interpreted the requirements differently than the developers? Does it happen often? Is it a problem? 5. Do you experience any problems due to the process of documenting requirements? 6. Do QA have any process that addresses quality requirements? If not, how are these handled?
Documentation
7. Which documents are produced by the projects? 8. Which documents do you use and how often do you use them? 9. What is the purpose of each document? Are all the documents and parts of the documents necessary? 10. Do the documents contain any duplicated information? 11. Is the quality of the documents good enough for you to be able to work efficiently? 12. Are you missing any documentation that could have been useful to you? 13. Do you feel QA in some way compensate for the lightweight requirements documentation? 14. Do you think test cases in some way replace requirements documentation? Does this approach yield any challenges and risks? 15. Do you feel that the requirements documentation is inconsistent with the actual functionality of the software platform? 16. Is some functionality in the software not specified? How are you notified about this functionality? Is it tested? 17. Are inconsistencies in the content and scope of some documents a problem?
Knowledge sharing
18. How would you describe the cooperation between QA and the development department? 19. How do you know what to test and what the expected behavior is? 20. How is QA notified when new functionality is developed? 21. Do you have to answer many questions about how the software should work? Who asks? Is it a problem? 22. Are there any typical questions? Could these questions be answered through documentation? 23. Would you save time through documenting the answers rather than to answer them yourself? Would the person asking the question save time? Overall, would the organization benefit from documenting more? 24. Would it help you if other employees documented more? Would you rather go through documentation than ask questions? Overall, would the organization benefit from documenting more? 25. Do the developers always remember functionality when you ask them about it? What do you do if they do not or if they are unavailable? 26. Does the amount of questions increase as the company grows?

27. Do you see any risks, with regards to scalability, to keep working the way the company does with requirements documentation? 28. How does your department introduce new employees? Do they experience any problems due to the amount of requirements documentation that is available? 29. Is it clear to QA when the software's functionality has been or will be changed? How is QA notified about the content of the change? 30. How does QA know what test cases to update after a change to the requirements? 31. Do you think any knowledge that is shared verbally needs to be documented? 32. Do you think it is a risk that much knowledge about the software platform is individual and not documented anywhere?
Other
33. What are your thoughts about the general quality of the software at release?
Extra
34. Do you experience that others complain about the documentation? 35. Do you experience any other problems relating to the amount of requirements documentation?

B.3 For product managers

Personal
1. What are your main duties and responsibilities? 2. What is your experience? For how long have you been working in your current role?
The requirements process
3. Draw the requirements process on the board based on how you perceive it. How do product managers decide on what functionality that is to be developed? Based on what is this done? Who are involved in the process? How are the requirements sent to the development teams? What is documented in the process? 4. How do product managers prioritize different tasks against each other? 5. How do product managers keep track of the features that are available in the platform? 6. Do you experience any problems due to the process of documenting requirements?
Interface towards development teams
7. How do product managers communicate with the development teams? 8. On what level are the requirements that the product managers send to the development teams? Is some information missing? Is it a problem? 9. Are there any required conditions that determine how the order should be written (e.g. level of detail)?

10. How do the projects refine high level requirements? Are the refined requirements stored anywhere and are they communicated back to the product managers?
11. How do the projects validate their output with the product managers?
12. Do the projects create prototypes in order to facilitate discussion with the product managers?

Documentation

13. Which documents are used in the contact between the product managers and the projects?
14. Which documents do you use and how often do you use them?
15. What is the purpose of each document? Are all the documents and parts of the documents necessary?
16. Do the documents contain any duplicated information?
17. Is the quality of the documents good enough to let everyone work efficiently?
18. Are you missing any documentation that could have been useful to you?
19. What requirements documentation (that is kept up-to-date) do product managers have? How do product managers keep track of what the products can do?

Knowledge sharing

20. How do you do when you need to know the functionality of the software? Would more documentation be useful in this regard?
21. Do you have to answer many questions about how the software is working? Who asks? Is it a problem?
22. Are there any typical questions? Could these questions be answered through documentation?
23. Would you save time through documenting the answers rather than to answer them yourself? Would the person asking the question save time? Overall, would the organization benefit from documenting more?
24. Would it help you if other employees documented more? Would you rather go through documentation than ask questions? Overall, would the organization benefit from documenting more?
25. Is it possible to find out why a feature behaves the way it does? If not, is this a problem? Do you need to know this?
26. Do you see any risks, with regards to scalability, to keep working the way the company does with requirements documentation?
27. How do product managers communicate with QA? How do QA know the correct behavior of the software when testing it?
28. How does your department introduce new employees? Do they experience any problems due to the amount of requirements documentation that is available?
29. How is knowledge about changes in the software's behavior communicated to stakeholders?
30. Do you think any knowledge that is shared verbally needs to be documented?

31. Do you think it is a risk that much knowledge about the software platform is individual and not documented anywhere?
Other
32. Do product managers have any process that addresses quality requirements? If not, how are these handled?
Extra
33. Do you experience that others complain about the documentation? 34. Is project specific documentation maintained? Why/why not? Do you need to maintain documentation to the extent that you do? 35. Do you experience any other problems relating to the amount of requirements documentation? 36. How is the platform organization affected by the product organization?

C. Organizational distribution of interviewees

The distribution of the interviewees is found in Figure 7 below.

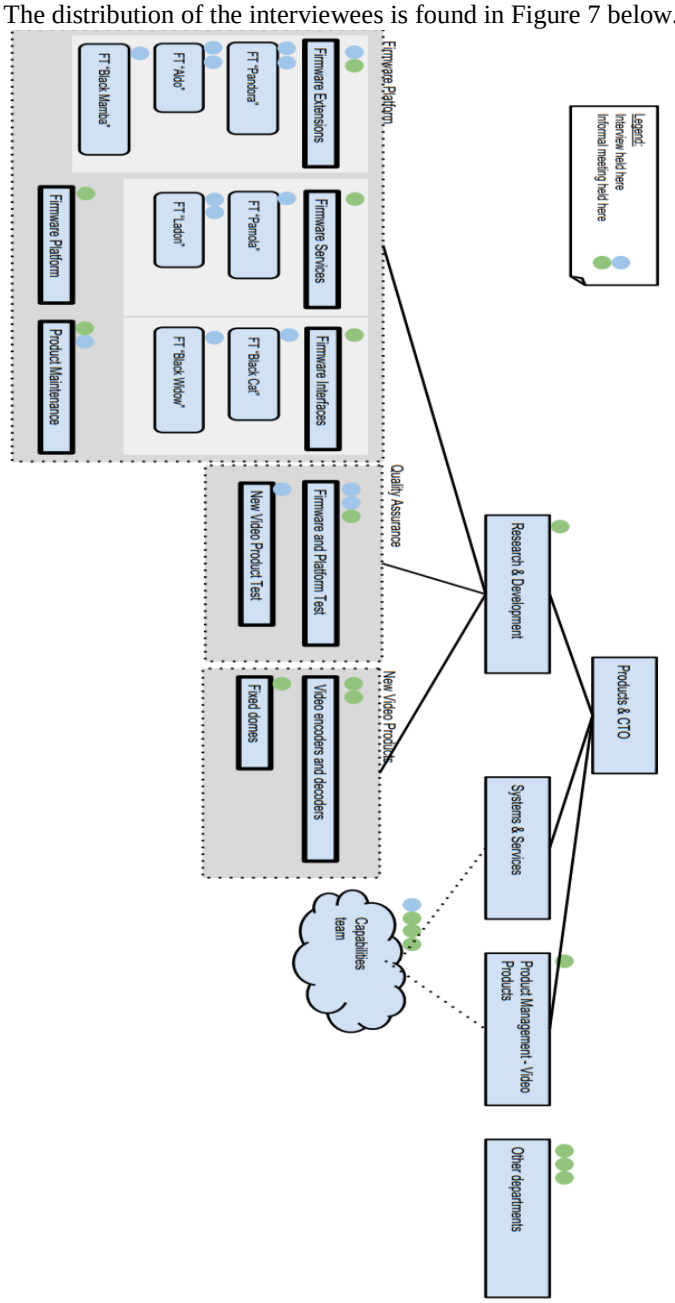

Figure 7. The distribution of where data was gathered from in the organization.

D. Assertions

In this section, the assertions (high level statements), extracted from the statements in the data analysis phase, are presented.

D.1 Tags from Firmware Platform

- Tests are in some way regarded as requirements
- Tests show what the software can do, rather than what it should do
- Some knowledge about the platform is individual, which is generally seen as a risk if people quit
- Axis has chosen to have “fleeting” requirements rather than a requirements database
- The order specifies how the software should work (requirements) whilst the PFD specifies how it is working
- Some developers think answering questions takes too much time, which in some cases has made them document answers to typical questions
- Questions (e.g. about functionality) can sometimes be answered by reading code or documents but there is always the possibility to ask someone. Asking someone seems to work quite well
- Functionality in previous releases can be found through flashing a camera and testing it
- People experience difficulties in knowing whether a functionality's behavior is correct or not
- Project specific documentation (e.g. order, SWO, backlog, PRS) are not updated after the project ends
- The PFD and the VFD lives on after the project ends and should always reflect the current functionality
- Integration notes describes which features that has been changed by a project and are sent to the LFP program upon release
- Documents are not stored in a common repository and are in general hard to find
- In some cases developers are not aware of how other departments work and what documents they use
- Developers write and perform unit and function tests during implementation
- Developers would like to have quick feedback on their implementation, e.g. through automatic testing
- Requirements are broken down using the order as a starting point

- Requirements are generally broken down through discussions in the team and with the orderer, facilitating knowledge sharing
- Even though the teams try to break down requirements through discussions, sometimes individuals start coding right away
- The development teams have started to use backlogs (owned by the orderers) containing prioritized activities
- Documentation on why certain decisions were taken has been nonexistent (which has been a problem), but some teams have started to document their decisions on their project site
- Why certain functionality exists and why it behaves in a certain way are common questions, which have to be answered by individuals since they are not documented
- The VFD is seen as a sort of requirements, which can aid the developers
- The VFD is often considered to be important and of good quality, but not everyone uses it
- The public API specifications are similar to the VFD and are sometimes used as a substitute
- The VFD is also used outside of development
- There is a separate group that decides on API changes
- The SWO is written based on the order, with the purpose to discuss and explore what areas/persons are affected by a project
- The SWO is used as a way of letting the orderer know what the project intends to do
- Communication between CBAs is facilitated by the work with the SWO, but does not seem to be dependent on it
- The SWO is considered important and well defined, but is not used a lot (and maybe not updated) after its review
- In practice, the SWO might not always be used
- You do not know what you want to change in the beginning of the project when the SWO is written, which makes it difficult to write the SWO
- Just talking to the relevant persons could probably replace the SWO, with the benefit that those persons can give early feedback (which reduces the risk of late changes)
- Asking questions is most of the time perceived to be better than reading documentation and leads to positive side-effects such as improved networking and knowledge spread
- The culture at Axis is open, meaning much knowledge is spread in a word-to-mouth fashion

- Many people stay in the company for a long time, e.g. making it easier to find experienced persons
- Function teams help spreading knowledge across the team, thus reducing the dependency on single persons
- Writing and maintaining documentation can be time consuming, e.g. due to reviews and re-reviews
- In general, developers want to minimize the amount of documentation
- Developers have different opinions about the usefulness of documentation
- Developers do not like to write documentation and would rather get to work (code)
- Basically all the documents are required due to the Axis project process
- The documentation process is not always strictly followed, e.g. not everyone updates the documents after changes
- Verbal communication is used as a complement to documents, reducing the dependence and need of documentation
- Developers want information to be up-to-date, correct and applicable, finding incorrect documentation to be more irritating than no documentation at all
- The need for communication could be reduced if developers could trust that documentation is up-to-date and contains what they need
- The order generally contains high level requirements
- The orders are in many cases vague and unclear
- Missing parts in the orders have sometimes halted the project progress
- Unclear orders increase dependence on the orderer
- An unclear order may give an unclear PRS
- When the order is unclear some effort should be put towards clarifying it early in the project
- An unclear order is not always a problem
- The order should contain a motivation, e.g. use case, for the requested functionality
- Project specific documentation is saved after a project has been closed, but the documentation is not maintained
- The orderer can be a product manager, a system architect or a CBA
- Wishes from other stakeholders are channeled through the orderer
- The team and its orderer works out what a project should do iteratively and in collaboration
- The orderer frequently participates in the project meetings, e.g. discussing and prioritizing the requirements

- The development team plans its work together with the orderer
- The development team breaks down the requirements in the order through discussions, both internally and with the orderer
- The project documents (SWO, PFD and PRS) and constant discussions are used to validate the project's ideas towards the orderer
- The project performs demos with the orderer with the purpose of getting early feedback on its work (e.g. catching changing requirements faster)
- Having demos more often means shorter time to feedback
- Demos does not assure that the orderer will not change his mind later anyways
- Demos may not be appropriate for some projects, e.g. in technical (non-visual) projects
- The project creates use cases by themselves or in cooperation with the orderer
- The project is very dependent on the orderer being available
- A lacking communication between Product Management and the project can lead to unnecessary rework
- The orderer does not always work closely with the project
- The orderer lacks a clear scope of what he/she wants, leading to changing requirements throughout the project
- When asking product specialists or product managers ”how should it work?”, a common answer is ”it should work as before”
- Product Management does not keep track of all details, leaving some of that responsibility to developers
- Technical writers base their work to some extent on documentation (CD/PFD/VFD)
- The PFD is a document describing/showing how functionality currently is working, on a relatively high level
- The PFD is used outside development, e.g. by QA and in the communication between orderer and project
- The PFD is reviewed by the team and other stakeholders
- The purpose/usefulness of the PFD is often not clear to developers, e.g. causing them to forget about it as well as complain about having to write it
- To some developers it is not clear what should be in the PFD or how it should be written
- The PFD might not be applicable for all projects
- The quality of some PFDs is lacking (e.g. outdated)

- The development team creates estimates on project duration, e.g. based on the requirements in the PRS
- It is difficult to create reliable estimates, due to changing requirements and difficulty in specifying the requirements up-front
- Incorrect estimates cause postponed integration dates, which is an issue when coordinating many teams' integrations
- Product Maintenance sometimes needs to ask the developers (CBAs and CBMs) for help when fixing CSTs
- Normally, Product Maintenance does not use the PFD or other documentation
- The growing platform makes it difficult for Product Maintenance to keep track of all the functionality, which means Product Maintenance tries to split up in smaller more specialized teams
- The project creates the PRS based on the order
- The PRS is reviewed by the orderer and other project stakeholders
- The PRS is often not used in the projects and has in many cases been replaced with a backlog
- In some projects the PRS was basically the same as the PFD and the VFD
- The PRS specifies requirements and the project deliverables
- It is pretty clear what the content of PRS should be
- In general, developers do not use the PRS during the project after it has been written and approved, since they already know what should be done at this point
- Developers do not see the use of creating the PRS (more than to pass a tollgate)
- If the PRS was more detailed developers could use it when writing tests
- The PRS is used for communicating what the project is/was supposed to do with stakeholders, e.g. the orderer
- The PRS is not always updated during the project, leading to people not trusting it and rather asking someone instead
- The PRS is used more during product development (NVP) than software development
- Each project team has dedicated QA resources who participate in team meetings and tests the team's output
- The dedicated QA resources that participate in team activities facilitates communication, e.g. of what/how to test something
- The team aids its QA resource with writing/reviewing test cases, e.g. in order to confirm that the test cases are in line with expected behavior
- Changes are communicated to QA through tickets and release notes

- The team's QA resource reviews the PFD and the VFD
- QA uses the PFD as a help when writing test cases, and might not even start writing test cases until it is done
- QA sometimes uses the PRS when creating test cases
- Sometimes QA are not notified of changes in the software
- It is pretty common that the development team gets incorrect tickets from QA because QA has misinterpreted the expected behavior of the software
- The values of the quality requirements (e.g. performance) are somewhat arbitrary and it is often unclear what they are based on
- There has often not been any quality requirements specified for a project, however, the development teams has started to measure performance to make sure it does not deteriorate over time
- QA is seen as the ones being responsible for checking quality aspects and making sure they do not drastically deteriorate
- If a quality related test case fails attention is directed towards it, although it might be decided to let it be as it is
- The orderer decides on quality requirements and are sometimes aided by the CBA in the process
- Generally the software quality is pretty high compared to other companies, but occasionally complaints has been raised by the customers about performance issues
- There are projects focused on quality aspects (e.g. scalability and performance)
- The orderer sometimes needs to change quality requirements since the initial ones were not feasible
- Performance differs greatly between different products
- Quality aspects has become increasingly important
- The CBM is responsible for the functionality of his code blocks, making him/her a primary source of information regarding functionality and changes to those code blocks
- The CBM does not always have detailed knowledge of his/her packages
- If the CBM is unavailable, there are other people who have the information
- The CBA has overall knowledge and responsibility of an architectural area, making him/her important as a coordinator and information source
- The CBAs mostly have discussions based on need, rather than regular meetings
- The CBA keep a frequent communication with NVP, since NVP talks to the CBA when they change/wonder something about his/her areas

- The project managers are responsible for the creation and maintenance of the PRS
- The CD is written (and owned) by product management and then iterated with the development team
- The CD describes the high-level characteristics of a feature
- The CD contains duplicated information from the PFD and VFD
- The CD might replace the PFD, but it also might not be used in the future

D.2 Tags from QA

- The fact that someone from QA is part of the development team works well, e.g. since it improves the communication between development and QA and also the quality of the test cases
- The TAM and the development team cooperates in the creation/updates of test cases and in the refinement of requirements
- The TAM feels that asking questions to developers is more useful than reading documentation and probably solves more issues
- Does not think it is possible to get the orderer to write better requirements and to get him to maintain the documentation, at least not after the product has been developed
- The orderer does not know in detail what they want until the end of the project, which causes them to solve the problem they want to solve rather than the solution to the problem
- Thinks it might be good that the requirements are quite poorly specified since it makes the developers and testers to try to understand what product managers really want. This makes them less dependent on product managers to micromanage them
- Wants to use web pages for specifying tests, making them readable for orderers who then can judge if they are correct or not (behavior)
- It is unclear whose responsibility it is to update the CD, resulting in it being outdated
- TAMs are in cooperation with developers writing tests that specify the intended use a feature, outside which Axis does not promise to provide support
- Sometimes think the content of the PFD is unclear
- At NVP, the PFD does not seem to be updated always (at least not on the project website). This is not necessarily needed but QA would at least want to be notified when changes has been made so they can update their test cases

- Missing PFDs can be a bigger problem than outdated PFDs, e.g. leading to late discoveries of functionality in testing or difficulties to discover inconsistencies in the web GUI
- People in QA have different opinions regarding the necessity of the PFD
- The abstraction level of the requirements in the PRS differs greatly, leaving a lot up to the individual's own interpretation
- The quality of the PRS depends on which project and project manager that has written it, but in general it is not very good, making QA's work inefficient
- The PRS might not be updated toward the end of a project and different projects are different good at updating it when changes has been done
- The perceived usefulness of the PRS differs among testers
- In agile projects user stories and use case has replaced the PRS, seemingly making the TAM's work easier
- Newly employed think better documentation will solve all problems, but more documentation can increase the risk for mismatches between documents and actual products, which are cumbersome to work out
- When documentation is lacking, discussions are held with the developers
- Face-to-face communication is generally better and more useful than reading documentation, but one tester that is not part of a development team feels it is not optimal
- Documentation is good for handshaking purposes and as tools for communication, but its usefulness is limited after it has been agreed upon. This means it should not be maintained
- Too much documentation might make people lose focus on what really matters (products, tests and human communication)
- Since QA find it difficult to know what functionality is supported in a certain product/firmware, more tests than necessary are often included leading to quite a lot of time being wasted on evaluating if a test case is applicable or not
- People outside QA also review the test cases in order to ensure requirements have been interpreted the same way
- Occasionally test cases are not correct, but that is not necessarily a problem. Instead it is being solved through tickets, which is just another way of communication that gets you quick answers
- Some people at QA only use the intranet's project pages (and not Git) in order to find PFDs, creating the risk that they use old PFDs
- It is hard to get an overview of the PFDs, which has created a wish to collect them all in one place and add traceability to where they came from

- Knowledgeable individuals are important and losing one will result in a knowledge loss, however it might be hard to solve this problem through documentation
- Usually several persons can answer a question, but when no developer knows the answer one of them may have to dig into the code to find the answer
- One NVP tester finds the traceability between tests and requirements lacking, saying it is hard to do due to e.g. missing/scattered documentation and inconsistent numbering of requirements
- Because many test cases cannot be mapped to requirements, those test cases become the de facto requirements
- It is common to have comparative quality requirements (not worse than x, better than y), which are perceived as vague. However, the opinions about whether this is good or not differs
- QA is starting to measure quality aspects in a database, in order to be able to monitor the development of different aspects over time
- There is a confusion regarding what is "right" when it comes to quality requirements
- QA feel they are made responsible for "creating" the quality requirements There have been some issues with quality aspects, e.g. time consuming for QA to "find out" what quality levels are ok and performance issues when more functionality have been added without scaling up hardware
- Some person in QA is worried about not having clear goals for quality aspects (feeling restricted to just measuring what the camera "can" do)
- QA discusses with developers and read documentation (e.g. PRS and PFD) when creating test cases for new functionality
- QA uses old test rounds, documentation (e.g. PRS, SWO, user manuals, product notes) and own judgment when deciding what tests to include in a test run
- The introduction of TAMs has improved the test cases without any additional documentation, likely due to the closer communication with the developers
- Currently QA does not develop unit tests, since they are more focused on testing at system level
- There is an opinion that tests and products are enough "requirements documentation" and that having a requirements document only produces more overhead, but everyone does not believe it
- Generally QA feels that its test cases are defining the requirements, at least in many cases

- Even though the company works in a test driven way, pure TDD is not feasible since the tests need to evolve alongside the code
- It is common to use benchmarking as a big requirement, e.g. product x should be as product y but better
- ”Benchmarking requirements” are currently difficult to test and might pose a serious scalability challenge in the future
- One person thinks that “benchmarking requirements” are a good thing, e.g. making it easier for a product manager to specify what he/she wants
- One tester feels that it will be hard to verify if the set of functionality that has been switched on in a product is the right one, when the software is used as an oracle
- There is an opinion that documented requirements should be temporary (for various reasons, e.g. reduced maintenance effort and avoidance of literal interpretations) and that the focus should be on the product as requirements
- Currently a feature list is being developed, which will be a collection of all the different features available in the platform. Each feature will likely be defined by use cases and corresponding test cases
- From the “big” feature list (being developed), each product is supposed to be able to generate information about its own features for those who want to know what the product is able to do
- Having automatically generated product information will make it easier to keep track of the difference between products (aiding “benchmarking requirements”), but then the software in the product must not be wrong since it in that case will be the “oracle”
- People will focus on doing things that benefit themselves rather than others. Getting people to actually do something that benefits someone else is hard when they have stuff to do that benefits themselves
- People feel uncomfortable and get angry at the orderers for changing their minds. However, the real problem (according to one person) is people’s attitude toward agile
- The culture at Axis is very open and encourages people to talk to each other (even between departments), thus facilitating close collaboration
- Some people feel uncomfortable when talking to others and would much rather have clear instructions on what to do
- The fact that requirements are specified in an unclear way does not necessarily have to be negative, since it makes people talk to each other and lets them be creative

- One risk of using verbal communication to communicate requirements that are not documented is that other testers needing the same information are not notified about it
- A requirements database or big requirements documents for each product would be useful to QA, but many are worried that it would cost too much to create and be burdensome to maintain
- QA feels they have to weigh up for a lack in specifications of requirements, since they have to create test cases anyways
- There has been issues because errors are found late, which has caused Axis to try to push testing "upwards" in the development process (so tests are executed earlier) e.g. through introducing TAMs. Still, sometimes a lot of development is done before QA can test
- Long feedback loops in development have been an issue, since developers do not have the code fresh in mind after it has been sent to QA and then back to them
- QA have been focusing on black-box testing, which has not been useful for the developers, but are now developing automated tests
- Automated tests will give the developers quicker feedback, since they can run the test cases by themselves, and also enables testing on a deeper level than QA usually does (e.g. testing APIs)
- NVP test finds it difficult to know what functionality there is at the firmware platform
- Different parts of the organization use different names for the same feature, which has caused confusion
- It is really important that engineers understand the bigger picture (business model, customer needs, etc.), which is reflected in the engineer career ladder
- The perception is that the quality of the software at release is good, e.g. due to rigorous testing
- The open climate makes it easier for new employees, although it can be difficult to know who to ask when wondering about functionality
- Would like product notes to be available earlier in the project since QA uses it when writing tests, but has met much resistance when asking for this

D.3 Tags from product managers/specialists

- The product managers create strategies for their own areas through prioritization of different ideas during the roadmap work. The prioritization is based on the viewpoint of the overall business strategy developed by management

- Some people expect very detailed requirements, e.g. R&D has historically wanted very clear orders
- Product managers prefer to express requirement in use case or problem form
- Orders from architects are probably more detailed than from product managers
- Orders from product managers for video products are more detailed
- The people in Product Management need to keep a good sync on what they are doing, since they have different opinions on quality priorities and changes to one part affects the whole platform
- The orderer does not want to specify details about behavior, e.g. because there is a risk that the team over implements the details and because he rather spends his time keeping track of the market needs
- CBAs can sometimes be orderers for a project
- One orderer prefers to work closely with the project in the beginning to better aid their understanding of the project, since the order is on a high level
- A high level order causes one product manager to have a greater dependence on an experienced project manager to break his requirements down
- One orderer feels that a high level order gives the project and its orderer some flexibility to change scope and puts the choice of solution into the team's hands
- Inexperienced teams who are used to more detailed orders have more issues with a high level order, resulting in greater dependency on the product manager
- The orderer follows projects in order to see if they have interpreted the order correctly, thus making sure that they are on the right track
- To reduce the risk of implementing the wrong thing, the project conducts workshops together with the orderer, writes a PAD which he looks at and conducts prestudies
- The SWO helps a project to get its initial questions down on paper, which the orderer then can follow up on to see the progress of the project
- Generally, the software produced in a project reflects the orderer's wishes
- One product specialist finds the PFD rather technical and hard to read
- One product specialist mainly uses mail/direct contact (but also the PFD) when uncovering the content of a release
- The PFD is not very useful for product managers who are not interested in the web interface (the customers of 80-90 % of all sold cameras do not use that interface)
- PFD and CD cover roughly the same needs

- QA base their tests to some extent on CD/PFD, but writing tests for everything in those documents can be hard
- The introduction of CD was not perfectly smooth, which has caused several things to be changed down the road
- The product managers will be the owners of the CDs
- Product Management sees the CD as important and useful, both as a knowledge base and as a 'sync' document
- The CD is seen as a replacement to both PRS and (at least parts of) the PFD by a product manager
- One of the uses of CD is to 'sync' Product Management with developers with testers, e.g. achieved through collaboration in writing the CD
- The PRS is too detailed for product managers to use, since it is hard to see the bigger picture from all the detailed requirements
- Product specialists have some contact with QA since they handle customer issues
- Product managers have a lot of communication with technical lead
- The orderer participates in some of the team's activities in order to follow their work and answer questions
- Internally, Product Management has meetings as well as informal communication
- When a CBA is the orderer, he/she syncs the project progress with the system architect
- CBMs are notified before developers make any bigger changes to their code blocks
- You want to ”force” the right people to talk with each other at the right time, exactly how you do it is not important. E.g. it can be done through writing and reviewing a document
- The organization is built around the idea of very frequent communication inside the function teams, but as limited communication between teams as possible (e.g. through architectural delimitations and APIs)
- The CBA have knowledge about the details in his area, how it works and why
- Different people want to work different ways, for example some project managers want more detailed orders – it is hard for product managers to know what level of requirements is appropriate
- Product managers want developers to think for themselves and understand the ”receiver's” (e.g. customers) situation in order to develop good software
- Developers are often not aware of all the areas where a document is used, causing them to not see the full purpose of the document

- New products have a more defined quality testing (mandatory quality tests) than new firmware/feature versions, indirectly also testing the firmware
- Other than the basic quality tests, there is no special process for managing quality requirements. Instead, questions about quality are handled from case to case based on the opinion of the product manager
- Measurement of performance have enabled following the development of performance over time, making those aspects more manageable e.g. catching bad performance trends easier
- The product manager does not see any tendencies for scalability issues in the processes, but he feels that the main risk is that things are documented for the sake of the process
- As the platform increases, it gets harder and harder to have people working broadly on it
- Scalability is to an extent handled through managing team size and structure (which also means managing the architectural structure), splitting teams whose area grows too large
- There is not one place where you can find detailed information on what a given firmware for a product actually can do
- It can be hard to find out the purpose (e.g. who requested it) of a certain functionality, but it would probably not be viable to document it
- It is hard to know what products use what firmware versions and which firmware versions that have which functionality
- Creating ”supported” use cases would make it easier to answer detailed customer questions, since they are ”on their own” if they go outside of those use cases
- Somehow being able to find out the purpose of a feature, why it was done, how important it is and what was agreed on is helpful in keeping track of what is the correct behavior and prevent QA from ”driving” requirements
- The quality of existing documentation is good, but there are some areas that are missing. This is being solved through writing documentation when changes are done in undocumented areas
- In general, the current process feels adequate with regards to documentation. The problem is rather that people does not follow it
- The projects have some freedom in what documentation they want to use, but there is a basic amount of ”process documentation” which is mandatory and may only be removed through an agreement with all users of a document
- Having a single process that suits many different teams is risky since the process elements may not be based on needs in all instances
- Agile is preferable for an orderer mainly since it gives flexibility

- Demos or CD are much more useful than a list of requirements to the product manager
- Estimates are rough at road map level and more certain once the project has gathered enough knowledge to give its own estimate
- Estimates are important in order to be able to choose the best activities from a business perspective. This also means that you can prioritize and do the most important things first (and possibly remove the less important things if the project gets delayed)
- The unit and function tests ”specify” the correct behavior to the developers