Software Requirements Division

[An Interview Study at Saab AB, Electronic Defence Systems] *

Emelie Tyvik
Department of Applied Information Technology
Forskningsgången 6
Gothenburg, Sweden
emelie.tyvik@gmail.com

ABSTRACT

Software requirements are a crucial part of software development. They are also part of the main reason why projects fail. The previous research in the area of software requirements has not been focused on the division of software requirements. This paper presents an interview study with such a focus. Five people working at Saab AB, Electronic Defence Systems, Operations Göteborg (Saab EDS) were interviewed about software requirements division and how they conduct such divisions at Saab EDS. The respondents answers were summarized and analysed. The analysis showed that Saab EDS divide software requirements into four levels; customer, system, subsystem and lower level. The software requirements division are conducted through group discussions. The basis of the division of software requirements is indicated to be based upon expert knowledge and that the software requirements division decisions are people dependent. The improvement that Saab EDS can proceed with, suggested by the paper author, is to take "software requirements division decision"-notes to be able to keep track of the reasoning behind the software requirements division decision.

General Terms

software requirements division, software requirements, software requirements engineering, software development

Keywords

requirements, division, development, software, industry, interviews, qualitative

1. INTRODUCTION

The software requirements process is considered a critical aspect of software development[25][1] as software requirements are the rationale for the software and the customer's needs.[23][16] With a strong increase in the IT sector since 2003 in Sweden[26] and inadequate software requirements being the largest problem when developing software and the main reason to failed projects[9], how can companies secure that projects do not fail and loose a large amount of money? One key is to shift the focus to requirements and realise their importance for the success of the project[9][15] and improving the requirements process can significantly improve the probability of the software project to succeed.[15]

Software requirements are a part of the software development process in the requirements elicitation and analysis step.[25] Wiegers states that there is "no universal definition of what a requirement is"[28], but there are many definitions of what a software requirement can be. Sommerville describes the difference in requirements definitions to everything from "a high level, abstract statement of a service that the system should provide or a constraint on the system"[25] to "it is a detailed, formal definition of a system function".[25]. To grasp the concept of what a requirement is, Aurum and Wohlin states that a requirement shows "what the system should do, rather than 'how' it should be done".[2] IEEE has published a definition of what a software requirement is and that definition will be used in this paper, since it is a standard and as Aurum and Wohlin states it is the one definition that is usually cited.[2] The definition is as follows:

> "(1) A condition or capability needed by a user to solve a problem or achieve an objective.
> (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
> (3) A documented representation of a condition or capability as in (1) or (2)."[18]

This paper will focus on specific aspects of software requirements: division of software requirements. The software requirements division process will be investigated in a real industry setting at Saab EDS, at the Operations Göteborg section, in order to investigate how industry reasons about software requirements division as practitioners.

The intended contribution of this research is:

- To increase the body of knowledge on how software requirements division is conducted in an industry setting.

*This paper is available through the IT University of Göteborg and the University of Gothenburg.

The author of this paper, Emelie Tyvik, is a student at the Software Engineering and Management program at the IT University of Göteborg. She has a Bachelor of Arts with a major in Sociology from the University of Borås.

The questions to be researched in this paper are:

1. How do Saab EDS divide software requirements into different levels of abstraction?
2. What is the software requirements division based upon at Saab EDS?
3. How is the validation and verification of software requirements effected by the division of software requirements in different levels of abstraction at Saab EDS?

1.1 Structure of Paper

This section will be followed by an overview of the previous research done in the software requirements area. After that the research approach will be described and how the method is applied in this paper. The research approach section will be followed by the data collection section, where the data found will be presented. The data will be analysed in the analysis section. The last section of the paper concludes the whole study and gives suggestions to further research.

2. RELATED RESEARCH

This chapter will contain a summary of the related research in the software requirements area and the position of this paper.

2.1 Software Requirements

The research done within the software requirements area includes everything from analysing the requirements in themselves and how they are interpreted to how one can monitor requirements with the use of code. Delugach's research is in one end of the spectrum with his paper "Specifying Multiple-Viewed Software Requirements With Conceptual Graphs" where he provides conceptual graphs that shows a general description of how one can interpret a requirement.[8] While on the other end Robinson suggest an implementation of requirements monitoring using the Java programming language.[22]

The Standish Group conducted a research in 1995 with the purpose of identifying the main reason to why projects fail. The conclusion of that study states that "incomplete requirements"[14] is the main reason.[14]

In the specific area of how software requirements are created, the focus has been on how to write them, not on which level they are specified on. The recommendations are for example that software requirements should be written in natural language, that the text can be supported with diagrams and equations[15], where text and graphical forms are useful to represent some software requirements.[28] Further details that should be noted are: use short and direct sentences, an active voice in the text, avoid synonyms, have a glossary, use the word 'shall' when writing software requirements. Another aspect that one should not use words that can cause ambiguity in the software requirements.[28] On a more abstract level it is stated that the software requirements should be complete and consistent.[25] The software requirements validation focuses on precisely those two aspects plus the accuracy of the software requirements.[15] Contradictory to all the above recommendations Wiegers state that it is impossible to specify the software requirements before building the desired system.[28]

When it comes to the division of software requirements the statements are of a more vague character, for example that different levels of software requirements are useful when dividing them[25], that the functional requirements should be specified on a level of detail that is suitable for the developers that shall implement them.[28] Furthermore, as a part of the requirements elicitation and analysis process, Sommerville expresses a step concerning software requirements division under the title "Requirements classification and organisation"[25]:

> "This activity takes the unstructured collection of requirements, group related requirements and organise them in coherent clusters."[25]

But there is no detailed description of how that division should be done. Terms such as requirements at organisational level, product level, and project level are used[2], but no details what characteristics a software requirement at the different levels consists of.

Aurum and Wohlin have also conducted a study concerning the decision-making process of requirements engineering. They found that through:

> "Studying the decision-making process in RE [requirements engineering] activities in more detail, it is possible to conduct an analysis of the RE [requirements engineering] process and its underlying decision-making processes."[1]

Aurum and Wohlin furthermore map the requirements engineering process with two decision-making models. They found that it is important to document the discussions and decisions that takes place and that one should do that on all levels in for example projects and organisations. This to enable a continuous tracking of the decisions so that they are actually made compliant with the goal of the business.[1] Aurum and Wohlin also suggests five steps to improve the requirements engineering activities:

> "1. Keep track of the decisions, their rationale and the effect that they have on the software product.
> 2. Identify the stakeholder involved in each requirements engineering step.
> 3. Identify the decision types, their actions and options.
> 4. Identify the knowledge needed for each decision phase.
> 5. Provide decision support tools."[1]

Aurum and Wohlin conclude that managing the requirements engineering will lead to developers being able effectively design a product that will meet the requirements that the stakeholder has."[1]

2.2 Case Studies

Several case studies in the software requirement area has also be conducted. Crow and Di Vito made in a study in 1998 where they compared four case studies on how one should formalise the requirements process for space shuttle software. Three of the case studies used a standardised way of verifying and specifying the requirements and the fourth used a state exploration method. They found that formal methods match the common requirements analysis processes and that formal methods can be beneficial to the product even if the method only is partially applied.[5]

Regnell et al. have also done a case study in the software requirements area. They investigated how a distributed process is proposed, observed and then evaluated in industry. The setting was in different marketing offices around the world. Regnell et al. found that if product management got charts on the prioritisation of software requirements from the stakeholders, it was possible to identify unforeseen differences in the stakeholders requirements prioritisations.[20]

Kotonya and Sommerville conducted a case study where they were involved in a project of developing a web-based library system for the United Kingdom Higher Education sector. They concluded that the use of viewpoints was effective when developing user requirements. Furthermore, when defining requirements one should use a multi-perspective technique to take into account the different stakeholder's requirements.[15]

Gorschek and Svahnberg performed a case study where they studied six companies who wanted to improve their requirements process. They found that through applying the assessment methods: model-based process assessment and inductive assessment, the companies had areas to improve in. Those companies with high domain knowledge in process assessment were the companies that could apply requirements process improvements to a greater extent than those companies with low domain knowledge.[11]

2.3 Paper Position

This paper is an interview study and will contribute to the present knowledge in the software requirements division area. There has not been any recent research at Saab EDS in this area and this paper aims to increase the domain knowledge at Saab EDS. The goal is; by increasing the domain knowledge at Saab EDS, they can in the future improve their software requirements division process to be able to even further enhance their products as suggested by Gorschek and Svahnberg report[11].

The theory in focus will be Aurum and Wohlins steps to improve the software requirement engineering as listed in section 2.1 Software Requirements.

3. RESEARCH APPROACH

This chapter will contain a description of the research approach used in this paper.

3.1 Interview Study

This study is a qualitative study with an interview method. The choice of a qualitative approach is based upon the research aim to get a deeper understanding of the division of software requirements at Saab EDS. According to Trochim using a qualitative approach enables one to achieve that - a deeper understanding of the object of interest. The positive aspects of using a qualitative method are that one will get a large amount of details based upon the viewpoints of the respondents. There are also drawbacks of using a qualitative approach, for example that it is hard to generalise the results.[27] In this research the aim is not to get a general picture of the software requirements division. The aim is to get a more deeper understanding of how the software requirements division is conducted at Saab EDS.

3.2 Finding Related Research

First the research started with a literature review on what has been researched in the software requirements area through a search with specified keywords in Google Scholar. The keywords are specified in Appendix A - Keywords. The first 4 pages (10 hits/page) were reviewed by reading abstracts, introductions and conclusions. Three books included in that search were ordered from other university libraries: one from the University West Library and two from the University of Borås Library.

3.3 Company Information

Saab AB develops systems for military defence and civil security, where Electronic Defence Systems is one of the business areas.[13] Electronic Defence Systems has as main focus on surveillance, threat detection and location, platform and force protections, and avionics systems. Saab EDS develops those solutions through the use of microwave and antenna technology.[12]

To be able to conduct the interviews, a review of Saab EDS's development process structure and background information was performed. A review of documents was also included, in order to be able to focus the questions and avoid asking unnecessary questions. Through that review the author of this paper identified that the development process at Saab EDS is of an iterative sort with a varying amount of cycles depending on the size of the development project. The different development stages are divided into smaller steps that results in a product and documentation of the work performed. This leads to that a large portion of the questions of the interviews was focused on the division of software requirements based upon a fact of the use of documents, since the software requirements division at Saab EDS is enforced by the use of documentation.

By getting a insight of the development process, a clearer understanding of whom to interview was developed. The mentioning of any classified details will be removed from the data collection since such information is company restricted. The mentioning of classified details such as product names, product areas and document numbers can reveal the identity of the respondents, thus breaking the anonymity agreement. Resulting in that such information has been removed and the respondents have been assigned with fictive names. This though did not effect the data collected and the conclusions drawn.

3.4 Interview

The interview questions were drafted based upon the research questions, related research and internal process and

requirement documents from Saab EDS. The models (listed in Appendix D - Division Models) were drafted based upon the internal Saab EDS documents on their process currently used, with the aim of being a list of believed to be exhaustive models. The respondents also had the option of drafting their own model. The use of open-ended questions was chosen to get the respondents to talk freely about the software requirements division. Using open-ended questions enables the researcher to get emerging data and will be able to develop themes of the object of interest.[4] The drawback with using open-ended questions is for example the large amount on non-answers. The so called non-answers, i.e. answers where the respondents does not answer the question,[21] will however still be reviewed to ensure that vital information is not overlooked. The interview questions are listed in Appendix C - Interview Questions. 25 main interview questions where stated. The questions were also discussed with the experts Ph.D. Thorvaldsson and Ph.D. Berling from Saab EDS, and supervisor Jonas Öberg from the IT University of Göteborg to ensure the appropriateness of the questions and alignment with the research aim. The interview questions were stated in an interview plan with instructions on how to conduct the interview and what to say to the respondents. This was done to ensure that the same information was presented to all the respondents.

An interview letter describing the research and the aim of the research was created and sent out the respondents. A description of the interview and research aim was also presented to the respondents before the interviews started. This is a part of the interview procedure that Bell enforces.[3]

A test interview was held to both test the questions and to get an estimation of the time it would take to conduct the real interviews. Doing a test interview is a good way to get feedback on the interview plan before executing the actual interviews from which you gather the research results.[3] After the test interview, the questions were narrowed down and some were removed due to not producing any valid material. This is a common results when executing a test interview.[3] The respondent Erik (fictive name) also had the chance to comment on the questions after the interview to improve the questions even further. The time was estimated to about one hour including preparation before the audio-recording started.

Five persons from Saab EDS were chosen from three product areas. The respondents either had a system level position or a subsystem level position in the development structure at Saab EDS. The fictive names of the respondents are: Bengt, Linus, Mattias, Peter, and Simon. The product areas are as well replaced with fictive names and they are named: product area 1, 2, and 3. The decision of choosing respondents from the different product areas was based upon the aspect that there could be a difference between the product areas and to have a diverse group of respondents. To ensure that the respondents actually worked with software requirements division, the respondents were chosen by Ph.D. Thorvaldsson due to the fact that she has a larger domain knowledge about Saab EDS, than the author of this report had. Ph.D. Thorvaldsson was the higher instance that Bell describes that one needs to ask to be able to get permission to conduct the interviews.[3] Five respondents was considered the optimum amount of respondents, based on an estimation of the amount of data each interview would generate and the time it would take to analyse it.

Before the interview the respondents were informed of their anonymity towards Saab EDS, and any other third party and that their name would be replaced with a fictive name in the report. Even though Ph.D. Thorvaldsson knows who the five persons plus the test respondent are she does not know who of them said exactly what. The respondents were both informed of this in the interview letter and verbally before the start of the interview. Choosing to have the respondents anonymous was based upon the fact that anonymity can enable answers that otherwise would be withheld due to dependency situation between the respondents and their employer. Having a anonymity agreement with the respondents is also an important aspect that one needs to have before the interviews according to Bell.[3]

The interviews were held in Swedish since it was the respondents native language and it was also a demand from Saab EDS, since the respondents could not be expected to be fluent in English. The data collected from the interviews that is presented in this paper has been translated from Swedish to English and any citations are also translated by the author of this paper.

The interviews were audio-recorded, so that the material collected during the interviews could be transcribed in a correct way. Ryen recommends to use an audio-recorder during interview to allow a correct reproduction of the collected material.[24]

The interviews took place at Saab EDS (two at Saab EDS' offices in Lackarebäck and three, plus the test interview, at the offices in Kallebäck) in a closed off room some distance away from the respondent's respective offices. This decision was based upon the fact that qualitative research takes place in what Creswell states as a natural setting and the researcher goes to the site where the phenomenon takes place to develop a higher level of detail.[4].

The length of the interviews were: Test interview: 43 min, Interview 1: 44 min, Interview 2: 25 min, Interview 3: 24 min, Interview 4: 36 min, Interview 5: 32 min. All the same main questions were asked to all the respondents.

The interviews were fully transcribed. The time spent on transcription was: Test interview: 4 h and 30 min, Interview 1: 5 h, Interview 2: 2 h, Interview 3: 2 h, Interview 4: 2 h and 45 min, Interview 5: 2 h and 15 min.

3.5 Analysis Method

The transcribed material was analysed with the focus on finding key points stated by the respondents, to find the sentences and material with high value and the material that answers the asked question. The transcribed material was also reviewed at least twice to ensure that the key points drawn matched the full picture of the answer.

4. DATA COLLECTION

This chapter will contain the collected data from the interviews. The data collection is structured based upon the

interview questions (stated in the Appendix C - Interview Questions) and the results are presented using summaries and quotations. The areas that will be gone through are: general questions, documentation, validation, verification and end questions. The respondents came from three product families: product area 1, 2, and 3 (fictive names). The following names used are fictive names, to ensure the anonymity of the respondents. From product area 1 a system level person was interviewed - Linus. From product area 2 two persons were interviewed; from system level - Mattias and subsystem level - Bengt. From product area 3 from system level - Simon, and from subsystem level - Peter, were interviewed.

4.1 General Questions

4.1.1 Division Procedure

All the respondents saw that there was a division of software requirements into different levels. Simon stated that product area 3 had customer requirements, system requirements in different levels and subsystem requirements in different levels. Peter, also from product area 3, stated customer requirements, system requirements, subsystem requirements and then another lower level of software requirements that focused on printed circuit cards and another that focused on components. Mattias stated system and subsystem requirements on product area 2. Bengt, also from product area 2, stated that they only had subsystem requirements. Linus, from product area 1, talked about customer requirements that can be any type of requirement; system requirements, subsystem requirements and radar requirements.

Peter and Simon both stated that they at product area 3 try to sit together with persons involved with the same level of development; the system developers sit together and the subsystem developers sit together when conducting the division of software requirements. Peter said that they had meetings. Simon stated it more as sitting together dividing the software requirements. Both stated that they tried to conduct the division in that way, but both expressed that sometimes they fail in doing so. Simon expressed:

> ”There is no idea to create requirements if the others below do not understand [them]”. (Authors translation)

Simon also expressed that in the end it is always the higher level of development that decides the software requirements division. Peter contradicted this with saying that the person responsible for the software decides what they will develop and that the subsystem people decides how they want to develop, and then they try to connect those two parts. Mattias from product area 2 stated that they also try to collaborate when dividing software requirements to get a cost efficient solution, but that in the end it is the system level that is responsible for ensuring that the requirements are divided from the system level to the subsystem level. Mattias also concluded that the subsystems write their own requirements for the subsystem level. Bengt, also from product area 2, focused on the collaboration aspect and he stated that they sit together when new requirements arise to find out which subsystems will be effected by the new requirements. Linus as well focused on the fact that the division is based upon many discussion, but sometimes the division takes place in a meeting and sometimes a person performs the division by themselves.

4.1.2 Division Discussions

The question of how the discussion actually looked like when it come to software requirements division, Mattias stated that the focus of discussion is on the end efficiency, the performance, the cost and the calendar time. Peter stated that software requirements discussion focused on that one had already chosen a platform or an interface that the rest must comply with. Simon also stated the arguments was often based on expert knowledge that only a single person only had. Another argument was also that a person could refer to a rule for a software requirements division, but the others could never find that rule even though they had searched for it. In the end one had to go according the hierarchy (system level dominant over subsystem level) to resolve some issues. Simon stated that there could be many discussions but that those discussions occurred more on the lower level, since the system level focused on what the system should manage. Even though there were many discussions they never turned into conflicts and that people that had been a long time in their post of responsibility had a lower amount of conflicts concerning the software requirements division. Bengt stated that it was hard to say anything general about the discussions, but in the end there is always a good will to solve any issues to get the best result. Linus saw that some of the arguments focused on calculation capacity, cost, maintainability or code. But generally there were many different types of arguments.

Mattias stated that he liked the way they had their discussion now, but saw that it was people dependent; if he knew the people involved it made it easier to find a good solution. Mattias also noted that in the end not everyone can be satisfied with everything and that there will always be someone that has to do more then someone else. Peter saw that they needed a bit more clarity but as soon as rules are written down one does not want to follow them. Simon saw that there has to be more room for discussions and today people talk too little with each other. Bengt was satisfied with how it was today and it all comes down to having the right persons involved. Linus was as well satisfied with how the discussions were conducted today. Generally all the respondents were positive to how the software division discussion were conducted.

When it came to the question of how the software requirements division process would look like if they could change anything, Mattias suggested the idea of having a document supporting the division. Peter would like to have a model to support the division. He suggested that for each project, in the beginning, the group would decided a specific division model to be used and that they would get time to educate themselves on that model and then use it. Bengt stated the following about the division of software requirements:

> ”Things like that [software requirements division] are really difficult to build into processes [...].” (Authors translation)

Bengt also stated that in the end it was all due to personal chemistry when it came to a critical division moment. Simon further continued on the need for room for discussions and also adding that a better collaboration needs to take place. He saw that the subsystem level often had too little time for such things. Linus focused more on that there was room for building better constructional specifications then seeing anything in the software division that needed change.

The respondents were also asked if it was something characteristical in the software requirements that determined the division. Peter did not see such a connection, neither did Simon. Bengt saw that, since subsystem level decides the division of software requirements themselves, it is not something characteristical of the actual software requirements, but the division is determined on the person's experience. As he concluded:

> "If one has a concept that shows to hold, of course you try to follow that [concept]." (Authors translation)

Mattias saw that it all came down to the higher division on subsystem level but one could not formulate a simple answer for it. Simon stated that it was not the software requirements in themselves that decided the division and stated that it was dependent on the background of the person dividing the software requirements. He continued with saying that often performance requirements was highly determining for the software requirements division since they would effect the whole system. Linus saw that characteristics concerning software requirements on a subsystem level focus on either construction or algorithms, and on a system level the focus was more on an overall picture of construction. Linus also pointed out that the division was based on a natural order that was already present in the development process and that software requirements division was supported by documents.

4.1.3 Division Model

All the respondents were shown six models on how software requirements could be divided down to be on different levels. The respondents were also given the choice to draw their own model. All the respondent stated that model four was the one used as a standard. Model four depicts the division process where a software requirement is presented as a customer requirement. That customer requirement is divided down to a system requirement and then split into two subsystem requirements. Model four is stated below. The other

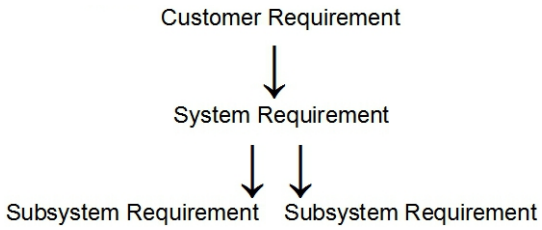

Figure 1: Model 4 - Software Requirements Division

five models can be seen in the Appendix D - Division Models. None of the respondents chose to draw their own model. Bengt and Mattias stated that for product area 2 they used model four. Mattias thought on of the other product areas, product area 3, used model two. Simon and Peter from product area 3 both stated that model four is used. Simon though saw that sometimes they use model three and it was dependent on that some software requirements could not be tested on the subsystem level and had to be tested on the system level. Peter also said that sometimes they use model one, and they use model two in a few cases if they for example buy an already made subsystem. Model three he has tried to use, but it created too much documentation, and that model six is used when they conduct internal development projects. Linus pointed out sometimes, new software requirements could arise on the subsystem level inside model four. No one of the respondents said that model five was used.

4.1.4 Legacy Systems

When it came to the question regarding if products and projects had software requirements that continue on to the next project in the product family (the next version of the same product) both Simon and Peter from product area 3 saw such a continuous process. Simon stated that they have a consistent software requirements structure. Peter also mentioned this by stating that since the platform is the same between projects many software requirements are the same. Peter stated that since the products they develop have a life time of at least 20 years but since software has a tendency not to live more than five years that when updating the software some previously made custom solutions can instead be bought, since the market has had time to adapt. Mattias, from product area 2, stated that 95 % of the software requirements are the same from one product to the next version of the product. But sometimes new requirements has to be created due to the fact that people either have put together many requirements into one requirement or that they were not formulated in the right way. Bengt and Linus both stated that most software requirements are the same from project to the next.

Then the respondents were asked if there was something in the software requirement division or the software requirement that determined if the software requirement was kept in the next product. Bengt saw that performance requirements often continued on, but that persons responsible for handling the requirements were not always the same. Mattias focused more on that from one version to next they tried to improve the software requirements and that it was all dependent on the unique requirements that the customer had. Peter saw that there was nothing in the division of the software requirements that determined if they were kept or not, since improvements are made constantly. Simon also saw that the software requirements changed from one version to the next, but the ones that often continues on are performance and functional software requirements. At the same time they also try to keep the subsystem parts and function blocks so they both can be re-used and sometimes also be implemented into other product families. They tried to keep the software structure, since they build upon a basis from the 1990's over time, to improve even further. Linus even went so far to say that all software requirements continue on to the next one but sometime new ones are added

to the consistent software requirement basis and that was not differentiated dependent any singular characteristic in the software requirement.

4.2 Documentation

4.2.1 Different Division Documents

The respondents were also asked concerning documentation of software requirements division. Linus stated that they had different specifications, but the software requirements division was specified inside those through a focus on different function blocks. Bengt described that they only had one requirements document, but also an electronic solution where all the requirements where listed. The division was also documented in this electronic tool. Mattias described earlier in the interview that the requirements were documented on different levels. Peter described that they have a model for document requirements and that they use a written document in a text format, which either is called functional specification or construction specification. He also described that they had a paper that he called 'division paper'. Simon described that they have customer requirements specifications, system requirements specification and subsystem requirements specification followed by a construction specification or design specification. Simon also described that they use a requirements metrics where they divide the requirements into different part of the system.

Mattias was not satisfied with the way the division was documented. He wanted that in the constructional specification it ought to be clearer what software requirement that was solved when a solution was applied. It was hard to trace the requirements up from the subsystem level to the system level. Peter did not want to document the division of software requirements and the documentation should only be applied in "really tricky cases" (Authors translation). Simon wanted to document the division in the way that it was conducted at the moment, but stated that the issues lies in that sometimes they do not follow the division process. He though pointed out that there was an improvement process that was on the way, where they are starting to be able to trace a software requirement even down to the single line of code. Simon stated that this was a "difficult area" (Authors translation). Bengt thinks it works really well and that people are clear with what works and it was easy for people to read up on the software requirements. Linus was also positive about how the division takes place, but saw that there was always a problem to draw the line between system and subsystem level. He thought that it was hard having different levels based upon the inheritance from the previous versions of the product, based upon that earlier all the software requirements were written by the system level but had now changed to that both system and subsystem level wrote the requirements. Linus concluded that in the end it came down the decision of how the software requirements were to be verified, but that all the division was done in a collaboration.

4.2.2 Receiver of the Division Document

When it came to if the division of software requirements are effected if the writer of the document knows who is the recipient of the software requirements document; all the respondents agreed that so was the case. Linus thought that it did and he really hoped that everyone knew who they were writing for and what the aim was of the writing. Bengt stated that it depended on the persons people are working with and when creating and writing a division document one always writes for a person that was lower down in the system development hierarchy. Mattias also thought it effected the writing but did not know if it applied to everyone else. Simon thought it effected but only to a little extent, but in the end it all came down to individuals and how they write. Peter thought that it effected, but the documents were seldom adopted to who that was going to read it, which he thought ought to be done.

If the software requirements division document was vital for them to be able to conduct their work Peter answered that it was not vital for him. Simon answered that he did not need the requirements in a document since he knew the product and thought that it depended on his experience. He also said newly employed persons would want all the requirements to be written down, but those who had worked at Saab EDS for 10-15 years only needed a meeting and then they could start developing, without documentation of the software requirements division. He concluded that there needs to be a balance of the amount of what was documented and what was not. Bengt said that he would be able to work without the documentation, but to get quality in the product and to be able to verify the product they needed to have documentation. Also Linus did not see any personal need for the division documentation, but saw that it was good to have a structure.

4.2.3 Detailed Descriptions Are Important

If there were aspects that could be removed in the software requirements division document, Mattias could not think of anything to be removed. Simon thought initially that there were things that could be removed, but ended up in the question stating that there was often to little information between the system and subsystem levels in the documentation. Peter thought that the requirement of actually being required to develop the documentation could be removed. Bengt did not find anything on the spot that could be removed. He focused his answer more on that when developing software requirements people ought to have a broader view and think of the whole system before developing specific software requirements. Linus thought that there was probably something that could be removed but did not give an example of what that could be.

In the aspect of what needs to be kept in the software requirements division document, Bengt stated that the requirements description must be kept. Mattias stated that the division of what subsystem should develop a certain aspect needed to be kept. Peter saw the importance of keeping the information intact of what platform the system had been developed for. Furthermore also that the customer requirements and the lowest level of detail needs to be kept. Simon discussed that it depended on the size of the company and if you are a larger company you need to have more detailed documentation. Linus thought that he could not answer that question, probably since he in the previous question stated that they are currently restructuring some aspects of their process.

4.2.4 Effects on Not Documenting in the Right Way

Concerning what the effects could be if the software requirements were not documented on the right level: Simon and Mattias saw that the software requirements would not be implemented. Bengt saw that the quality of the product would become faulty. Peter did not state the same concern and described that it would sooner or later be implemented, if the software requirement was important. Linus stated that there would be faults in the construction or functions of the system.

If software requirements would be documented in the wrong document Bengt did not see that as an issue, since the software requirements would eventually be found anyway. Simon saw that the people would not know that certain aspects of the product existed. Mattias also saw that the development would become flawed. Linus listed that a function would be missed, that the product would not be verified or that the product would be flawed in the realisation. Peter saw a future flaw risk that maintenance would become flawed if the documentation was not consistent with the system that had been developed.

4.3 Validation

4.3.1 Division Effects Validation

Concerning the connection of software requirements division with the validation of software requirements, Peter said it effected, since the validation was very focused on the different levels of the product. Simon did not initially think it effected since all the involved from different levels were present in the validation, and further on in the interview he stated that they are connected since different levels are sometimes validated separately. He continued on by saying that some software requirements could only be validated together with another function block and then it was hard to validate a certain requirement if it was not validated with that function block. Linus described it as that the validation was effected by the division since the software requirements were grouped together in function blocks. Mattias saw it as an administrative issue, since sometimes many subsystems were validated jointly, but not as a development issue. Bengt stated that they were connected, since sometimes complicated software requirements descriptions made the validation hard to perform.

4.3.2 Getting the Context

On the question if they wanted to change something in the software requirement validation none of the respondents stated anything concerning the division of the software requirements. Mattias, Bengt and Simon were satisfied with the way the validation was conducted at present (by the usage of validation meetings). Peter wanted more room for discussion and that the documents up for validation ought to be combined with all the other documents so that the validation aim instead was to validate a context and not details in the document. Linus saw that there was room for improvement in getting a better overview of the above effected software requirements and to ensure that it really was the customer requirements that were realised in the function that was validated.

If there was any other information needed about the software requirements that needed to be present to be able to validate the software requirements, Mattias did not see any need for such information. Bengt stated that sometimes one needed information about the context of the requirement, even though that the software requirements ought to be independent. The context Peter stated was always needed to validate the software requirement. Linus also focused on context but thought the need for a know-how of the level above effecting requirements. Simon had not seen any such information concerning software requirements and did not see any need for that.

4.4 Verification

4.4.1 Build to Verify

Concerning the verification of software requirements and the connection with the software requirements division, four of the respondents Bengt, Linus, Peter and Simon all said that it effected. Mattias did not think so, but as he described it he was not involved in the aspect of verifying software. Bengt even stressed that the software requirements to a large extent were connected to verifying, since they were written with the aspect of being verifiable. Simon also saw that the division of software requirements was effected by the aspect of verifying them, since the verification and testing sometimes combined testing methods that were high in cost, to be able verify more aspects at the same time, even though the detail might not come from the same part of the system. There was also a focus on trying to get verification as soon as possible since the feedback would come earlier, if the tests were done at an earlier stage. Linus described that for example if algorithms were often tested in constructional tests and what is written in the software requirement specification was for example verified through a subsystem verification instead.

On the question if the respondents liked the way they verify now, concerning the aspect of software requirements division, Peter wanted them to be built upon experienced people. Through that one could both remove unnecessary software requirements and also verify software requirements that had been missed in the documentation. He also stated that one could not write a process for that, since it comes from routine. Simon thought that they needed more structure concerning verification and that a good way to go would be to use more software to help in that aspect. Furthermore, Simon stated that there ought to be more demands on the designers, so that they could catch more errors there. Bengt thought it would be a good idea to involve the verifiers more, to look at the software requirements and conclude if the software requirements actually are verifiable. Linus thought that they could improve in the constructional tests to find more faults on the lower levels, instead of catching them on a higher level. Mattias did not have any opinion about the question, as he stated that he was not involved in the verification.

Concerning if there was any information needed about software requirements division to be able to verify, Bengt did not see that information was needed. Simon stated that when verifying a requirement, a verification specification was used together with a requirements metrics. Peter saw that sometimes certain requirements were just submitted to the subsystem level as information, but were not going to be developed there. He questioned that one should submit in-

formation in that way. Linus wanted more background information, the thought behind the software requirement, so that people could make even better verifications than just sitting ticking of requirements that had been tested. Mattias, even though not involved in verification, stated that often the verifiers were not concerned with background information.

4.5 End Questions

4.5.1 Effects on Not Dividing in the Right Way

As some final questions the question of what the respondents could see would be the results if the software requirements were not divided into the right level. Mattias focused on the cost aspect and stated that it would become expensive and take longer time. Peter saw that the problem would be pushed upwards and create extra work. Simon stated that the software requirements would not be developed and he saw that the cause would be the lack of system knowledge, if software requirements were not divided on the right level. Bengt saw that the end product would be of poor quality. Linus pointed out that the end product would not be what you expected.

4.5.2 Alternative Methods

Concerning if the respondents wanted another way in which information was documented about how the system would function, Bengt said that they had used use-cases and that it was an internal requirement on subsystem level to use such, but in the end not all requirements were suited for a use-case description. Simon wanted a larger use of pictures in the descriptions, but not an overuse of diagrams or models. Peter saw that it would be nice to try another approach but a prerequisite had to be that one then needed a solid group to work with. Mattias wanted to see another method, but saw a problem with applying that at their work place, since their products were very unique and used a large amount of performance requirements. Linus stated that he thought model-based development would be a good way to develop, since it would give a better overall picture of the product.

5. ANALYSIS

Here follows an analysis of the data collected from the five interviews. The analysis is structured through the questions asked during the interviews and stated in the following categories: general questions, documentation, validation, verification and end questions.

5.1 General Questions

5.1.1 Division Procedure

Sommerville describes that there is a division of software requirements in software development[25], such a division is also conducted at Saab EDS. All the respondents identified that there were different levels of software requirements and the joint one was subsystem level requirements. An interesting aspect was also that in the labels of customer, system, subsystem and lower levels that there were levels inside those levels as well. No respondent described those in between levels in detail so there is room for further investigation of what those levels could be identified as. The terms of organisational, project and product level as described by Aurum och Wohlin[2] was not mentioned by the respondents. This can either indicate that those levels does not exist or that they do exist but under other names. Since the question was not formulated to explicitly ask about those labels it is also hard to rule out their existence, but only identify that the respondents did not mention them during the interviews.

All of the respondents stated that the division took place through a group of people having a discussion. But there was sometimes a deviation from that and as Mattias stated, in the end it was always the system level people that was responsible for the division from the system level to the subsystem level. None of the respondents mentioned that they took notes during those discussions. Aurum and Wohlin suggest taking notes is a good way to improve the software requirements engineering process.[1]

5.1.2 Division Discussions

When it comes to how the discussion about the division of software requirements looks like the respondents focused on two aspects. One of what the areas of argumentations were about, and the other was what caused the division. The areas of discussions were: end efficiency, performance, cost (mentioned by two), calendar time, platform and interface, calculation capability and maintainability.

One respondent stated a specific reason behind the decision of software requirements division. He focused on that it was based upon expert knowledge. Lyytinen and Robey state that an important "source of knowledge is internal"[17], i.e. that knowledge that comes from companie's own experiences. They even further state that it is more important and strategical than external information. Both due to that internal information can come from anything from informal communication to formal analyses of the experiences that has been gathered. But also to its low cost compared to external information.[17] Concluding that:

> "[...] internal learning can more often produce competitive advantages because such lessons may be concealed from other companies."[17]

The author of this paper see that for being able to handle such division, the expertise and experience knowledge that is required to conduct a successful division of software requirements, needs to value in a good way. As Gorsheck and Svahnberg reasons about domain knowledge being the foundation of improving the software requirements process[11], the author see experience as a foundation for being able to create a good division of software requirements. This is also visible when Mattias stated that their division discussions were people dependent. The aspect Bengt also pointed out as a reason for the good discussions, a further example that the discussions are people dependent. Curtis et al. states as one of the major aspects of what must be supported in software development. It is that broad communication that is needed to integrate people.[6] Hence one should support the division discussions that take place at Saab EDS according to all the respondents. As two respondents stated, the software requirements process is people dependent - hence not only dependent on one person. The author see that as a support for Curtis et al.'s theory of developing large systems projects is a team effort and that talented people operates

inside those teams. Hence also seeing that all levels of the software development must be seen as a larger social and organisational process.[6] Paulk et al. also states that for long term success one should not rely on sole individuals, but instead focus on[19]:

> "[...] building a process infrastructure of effective software engineering and management practices".[19]

One improvement aspect, that none of the respondents mentioned, is to take notes during the division discussions to keep track of how the reasoning behind the software requirements division went. This to be able to return in the future to that decision and be able to make a new well founded software requirements division decisions. This improvement suggestion is based upon Aurum and Wohlin's article "The fundamental nature of requirements engineering activities as a decision-making process", where they suggest five steps to improve the software requirements process. The above mentioned suggestion is a part of the first step for improving further, where one should keep track of the decision rational.[1]

Concerning if the respondents wished to change something in the software requirement process, all the respondents stated different solutions: having a division document, project specific models, more discussions. One respondent stated that the software requirements could not be built into the process, that it all came down to the people chemistry. That response shows what Aurum and Wohlin describes as that important decisions in software industry are subjective ones. But as Aurum and Wohlin also points out that there is a risk with such decisions and that precautions should be taken against such risk consequences.[1] The author supports the freedom of having such discussions and decision-making, but to minimise the risk, as Aurum and Wohlin mention with such a subjective decision[1], with the use of a formal method, with the support of Crow and Di Vito's conclusions of a formal method that is partially applied can still be beneficial for the product.[5] One should also remember that the software requirements process is dependent on what Sommerville calls 'local factors' where the expertise of the employees is one aspect (the system developed and standards that are used are the two other mentioned).[25]

All the interview respondents except one saw that the software requirements division was not dependent on a characteristic in the software requirement. One respondent pointed out that division was based upon the experience of the people involved in the software requirements division. Yet again supporting the previous analysis of people dependency and expert and experience knowledge. That analysis is enforced by that one respondent focusing on people background dependency. One respondent even took it so far as to say that their software requirements division follows a natural order already present in the software development process. The same respondent also stated that the software requirements division was already supported by documents, the same aspect another respondent from another product area sought as a change needed in the software requirements division. Here a documentation, as in a formal specification Crow and Di Vito sees it, is a good starting point, since if changes to the software requirements are introduced it will give the best return of investment.[5]

5.1.3 Division Model

All the respondents chose model four (See Appendix D - Division Models) as the division process of software requirements. The respondents also had the option to draw their own, but none of the respondents did so. Model four is also supported by Saab EDS's internal documents - hence the respondents state that they follow the same software requirements division process specified by Saab EDS. But the respondents also stated that sometimes they deviate from that process. The deviation reasons were that sometimes software requirements that was on the system level had to be divided down to a subsystem level since it was only there they were verifiable. Another reason was if the software was purchased from an external party. This indicates the fact that a process is a good way to support the present methods, even though they are only partially applied.[5]

5.1.4 Legacy Systems

All the respondents identified the fact that many of the software requirements are continued from one version to the next in the product family. Davis et al. states that to be able to reuse such documented software requirements, the sentences, paragraphs and sections must be easy to adapt to the next version.[7] Again pointing out a potential for Saab EDS to look deeper into when according to the respondents such a large amount of software requirements are being reused.

Two respondents saw that it was something specific in the software requirements division that made them continue on to the next product in the product family. They identified performance software requirements as those that continued on. One respondent earlier identified that Saab EDS develops products that at least has a life time of 20 years. This life time of 20 years is common for military systems according to Sommerville and classifies those systems as legacy systems since they have an old basis that is built upon.[25] Hence supporting the fact that many of the software requirements continue on at Saab EDS, not based upon the division, but instead on the basis consistency of the system. Sommerville also supports the continuous process of improvement that one respondent mentioned in the interview since in legacy systems that the basis is improved with new software requirements.[25]

5.2 Documentation

5.2.1 Different Division Documents

The documentation of software requirements was mentioned early by all the respondents, when they talked about the software requirements division process, before they got specific questions about documentation. The respondents mentioned a couple of different documentation places for the software requirements division:

- It was done inside other software requirements documentation.
- It was listed in an electronic solution.

- Used a written document with a list in text format.
- It was stated in a division metrics.

This gives a diverse picture of how the software requirements division is documented at Saab EDS. Since processes are "fundamental to human activities"[15] and since processes are dependent on the changes that take place in the organisation[15] and furthermore because of the process is essential[28], a solution the author see is to join those different approaches; to do as Wiegers suggests: give all the involved in the different projects a chance to see the effects of their different views and methods through a team-building activity to improve the understanding for software requirements.[28] Through that it would also improve the way they all document the software requirements division and then if they choose to change to having one way or to have different ways would be up to them. But at least as Wiegers states it will increase the knowledge about the different views[28] - hence the view that all of them have about how the software requirements division should be documented.

Concerning if the respondent were satisfied in the way the software requirements division was, a diverse picture emerged. The respondents expressed:

- Was not satisfied with the way the division were documented.
- Did not want a division document (with some exceptions).
- Wanted to continue in they way they do now.
- Thought it worked really well as it was done now.
- Positive, but could identify some difficulties.

Having such a diverse group of opinions gives that a difference in satisfaction regarding how the software requirements division is conducted is present at Saab EDS. This indicates an improvement area to look deeper into, if one wants to strive to get all involved towards the same satisfaction level of how the software requirements division is documented.

Comparing the above answers with the question about if they could wish for changes in the division process, some differences were visible. Mattias still stuck to his idea of having documents supporting the division. Peter, that did not want any documentation, with some exceptions, did not remention the idea of model-based method that he suggested earlier in the interview. Simon did not mention anything about documents in the earlier question, but emphasised a need for discussion and a better collaboration. This emphasis was not mentioned in the connection to the document question. Bengt earlier pointed out that the division was people dependent and in the later answer he mentioned the people aspect, but not as clearly, as he also weaved in that it is easy for people to read up on software requirements. Linus continued to not see the software requirements division as an issue. Only two of the respondents, Mattias and Linus, continued to express similar answers concerning the two questions. This might be dependent on the fact that the answers from the respondents became more extensive the longer the interview proceeded, and also giving the respondents more time to reflect, while they were being interviewed.

5.2.2 Receiver of the Division Document

Every respondent thought that the software requirements division was effected if the writer knew who they wrote for. This yet again points to the people dependency aspect of software requirements division mentioned earlier in this analysis.

None of the respondents saw their work as dependent on the software requirements division. Simon answer indicates the experience aspect since he stated that people at Saab EDS that has worked there for 10-15 years only needs to sit down in a meeting to be able to start to develop. Bengt's answer indicates an aspect; if one wanted quality in the product and to be able to verify the product one needed to have documentation. Linus also stated that one needed documentation to have a good structure. These aspects can indicate that people do not need documentation to develop software systems, but that they need experience and a place to communicate with each other. As Faraj and Sproull states:

> "[...] software development is knowledge work, its most important resources is expertise."[10]

5.2.3 Detailed Descriptions Are Important

Concerning if the respondents wanted to remove anything in the software requirements division document, the following results was concluded:

- Nothing could be removed.
- Could not suggest anything for removal.
- There was something that could be removed. (Did not specify what could be removed.)
- Initially thought that there was thing to remove, but ended up in that there was too little information in the document.
- Wanted to remove the need to develop the software requirements division document.

All these answers show a spectrum of answers from the respondents concerning the removal of aspects in the software requirements division document. Due to this diversity of answers the author can only see that at least the respondents have different opinions and that no joint picture can be seen in this question.

Three of the five respondents had an opinion of aspects that needed to be kept. The suggestions were the software requirements description, the division of what system that should develop the aspect, platform information, customer and low level requirements. One respondent saw that a larger company needs a detailed documentation. One stated that he could not answer the question. The suggested aspects to be kept, if all joined would give a document with a

large amount of information, compared to what Simon answered in the previous question that there today was too little information in the division of software requirements in the system and subsystem level.

The aspect of what could happen if a software requirement would not be documented on the right level, four of the five respondents expressed a negative view - such as that the software requirement would not be implemented, faults and faulty product quality would occur. One respondent had a more optimistic view, where he saw that sooner or later if the software requirements would be implemented if they were important enough. The majority of the respondents sees the importance of documenting the software requirements division on the right level. This can be seen in that none of the respondents needed the software requirements division to conduct their work. The question arises what is it then that needs to be in place to both secure the right level of documentation of software requirements and something one needs to have to conduct ones work. Here the author would like to return to the aspect of the use of discussion, where Curtis et al. see the broad communication will support the software development.[6] The author of this paper see that as the respondents expresses that there is a need for documentation, but these documents needs to be supported by discussions.

5.2.4 *Effects on Not Documenting in the Right Way*

If the software requirements would be placed in the wrong document, expressed by four of the five respondents the following risks were mentioned: that certain aspects would not be known to people, functions would be missed, maintenance flaws, inconsistency in the document, and flawed realisation in the product. Again the respondents showed that a miss in the division of software requirements can cause huge negative effects concerning the product. A finger of warning ought to be raised to secure that such flaws should to be minimised. One should remember the importance of documenting software requirements since it "[...] is the official statement of what the system developers should implement."[25]

5.3 Validation

5.3.1 *Division Effects Validation*

In the aspect of identifying the connection of software requirements division and validation, four of the five respondents stated that they are connected. Where the validation is effected by which level of software requirement and which software requirements that are validated at a certain point. Sometimes some software requirements are validated separately and sometimes they are grouped together. One respondent only saw it as an administrative issue and not as a development issue. Sommerville emphasises the importance of software requirements validation, since if there are flaws in the requirements document it would "[...] lead to extensive rework costs"[25], since it is cheaper to fix a software requirement issue in a document than having to re-design the system or fix code errors.[25] This will be returned to in the section 5.5 End Questions.

5.3.2 *Getting the Context*

Concerning if the respondents wanted to change anything regarding the validation process, none of the respondents stated the software requirements division. However, to improve even further, Peter suggested more discussions and Linus wanted a better overview of the software requirements. This implies a general satisfaction amongst the respondents that they like how the validation process is conducted today. Here it was another suggested more discussions, from a respondent who had not earlier in the interview expressed a need for having more communication between the employees through discussions.

The information questions about validation, three of the five respondents saw a need to get the context of the software requirements to be able to validate the software requirement. Two did not see any need for any other information that needed to be present. According to Sommerville one needs to look at the consistency of the software requirements. To be able to compare the software requirements so that they do not conflict with each other.[25]

5.4 Verification

5.4.1 *Build to Verify*

The connection between software requirements verification and software requirements division was strongly identified by all the respondents that were involved in the verification (four out of five respondents). One respondent described it as the software requirements were constructed with the aspects of being verifiable, which is one of the foundations that is identifies during the validation process. As Sommerville states it[25]:

> "A general principle of requirements engineering [...] is that requirements should be testable."[25]

One respondent described that the division of software requirements were effected due to the verification, since Saab EDS combines testing methods that are high in cost when testing different parts of the system. The focus was also to verify as soon as possible to get the feedback earlier in the development process. Here it is important to realise that there are software requirements that are hard to verify. Those are: ambiguous, undecidiable and not worth the cost.[7] To fight those problems of having hard to verify software requirements one needs to have experienced testers that can review those software requirements.[7] Yet again pointing out the importance of having experienced people in the organisation as analysed earlier in the sections 5.1 General Questions and 4.2 Documentation. This aspect is also brought up by Peter in the next question asked, if they liked how the verification is done now, where he points out that he wants them to build upon experienced people. And he yet again states what Bengt expressed earlier that one can not build a process for that. Peter also states that is based upon routine. Simon wanted more structure in the verification. Bengt wanted to involve the verifiers more in the software requirements. Linus wanted to improve in the lower levels, to find more faults in the lower levels instead of higher up. These three above suggests a desire for more structure, involvement, and room for improvement from the respondents concerning verification.

5.5 End Questions

5.5.1 *Effects on Not Dividing in the Right Way*

The aspect of what would be the results if software requirements were not divided into the right level was partly mentioned by the respondents in the section 4.2.4 Effects on Not Documenting in the Right Way. In the End Question section (See section 4.5 End Questions.) all the respondents expressed that there would be negative consequences. Mattias pointed to the cost effect and an increase in the development cost and as mentioned in section 5.2 Documentation, increased cost will according to Sommerville be the results of developing faulty software requirements.[25] Here Simon returned to the aspect mentioned of Gorschek and Svahnberg of domain knowledge, where they state that those companies with higher domain knowledge where those that could improve to a higher extent, than companies with low domain knowledge.[11] Simon describes that the lack of system knowledge would lead to software requirements not being divided in the right way.

5.5.2 *Alternative Methods*

The interviews were finished with asking a broad question if anyone of the respondents wanted another way of documenting the information regarding how the system should function. They suggested:

- Use use-cases to a lager extent.
- A larger use of pictures.
- Try another development approach. (Stated by two respondents.)
- Model-based development.

This shows that the respondents have a creative side and that they can see solutions to how Saab EDS should improve further.

6. CONCLUSION

This chapter contains the conclusions of this study.

A software requirements division is present at Saab EDS. The respondents stated a usage of the labels: customer, system, subsystem and lower levels for the division of software requirements. The division of software requirements takes place in a setting of a group of people having division discussions. The discussions often focus on two aspects: the way people discussed and what arguments to how one should proceed and what caused the particular division. The basis for the division discussions were mentioned by one respondent where he focused on the aspect of expert knowledge. That could indicate that one needs to value the expert knowledge and experience to be able to create a good division of software requirements. Another respondent followed that by stating that it was dependent on the background of the people involved. One improvement aspect, that none of the respondents mentioned, is to take notes during the division discussions to keep track of how the reasoning behind went. This improvement suggestion is based upon Aurum and Wohlin's article "The fundamental nature of requirements engineering activities as a decision-making process"[1], where they suggest five steps to improve the software requirements process. The above mentioned suggestion is a part of the first step for improving further.[1]

The respondents also identified that there is a connection between the division of software requirements and validation, where the validation is effected by which level of software requirement and which software requirement that is validated at a certain point. Three of the respondents stated a need for context of the software requirements to be able to validate them.

The connection of software requirements division and verification was strongly identified, where even the software requirements were constructed for the intent to be verifiable. The software requirements division is connected to when the software requirement is verified, since some software requirements are high in cost to verify. Where even if the software requirement was not optimal to develop on an a specific level it was still verified on the most cost effective level, hence moving the software requirement in the development structure based on the verification cost.

Since many of the software requirements are based upon discussions, a way to improve for Saab EDS would be to take more notes during those discussion, hence to apply the best-practice suggested by Aurum and Wohlin[1] to improve the software requirements division even further by keeping track of the software requirements division. Saab EDS could also focus on investigating further how the software requirements division process actually takes place, and look deeper into how the division discussions are executed, since that has been proven to be how the software requirements division process is described by the respondents in this research.

The main conclusions based upon the research questions in this paper are:

- The software requirements division is done in group discussions.
- The software requirements division is indicated to be built upon expert knowledge and is people dependent.
- Taking software requirements division decision-notes can help keeping track of the reasoning behind the software requirements division decision.

6.1 Future Research

To further research how the software requirements division takes place in industry, one could conduct an observational study to further verify that the process of software requirements divisions actually takes place through discussions or if they are conducted in a another way.

Another future research suggestion could be to investigate how Aurum and Wohlins five steps of improving the software requirements engineering[1] would effect the software development if implemented. To goal could be to verify if those five steps will give a positive outcome for the development of software.

7. ACKNOWLEDGEMENTS

The author of this paper would like to thank the following:

- Bengt, Erik, Linus, Mattias, Peter, and Simon (fictive names) - For sharing their experiences and knowledge.
- Ph.D. Karin Thorvaldsson - For her support in this research and expert knowledge in the software requirements area.
- Jonas Öberg - For his encouragement and guidance in academic writing.
- Ph D. Tomas Berling - For his expert knowledge in how Saab EDS functions and the on-site guidance.
- Saab EDS and involved employees - For the opportunity to conduct this research and their encouragement.
- IT University of Göteborg - For the three years at the Software Engineering and Management program.
- Family and friends - For their support in this research and in the author of this paper's academic studies.

8. REFERENCES

[1] A. Aurum and C. Wohlin. The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, 45(14):945–954, 2003.

[2] A. Aurum and C. Wohlin, editors. *Engineering and Managing Software Requirements*, chapter 1. Requirements Engineering: Setting the Context, pages 1–15. Springer-Verlag Berlin, 2005.

[3] J. Bell. *Introduktion till forksningsmetodik*. Studentlitteratur, 2nd edition, 1995.

[4] J.W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Method Approaches*. Sage Publications, Inc., 2002.

[5] J. Crow and B. Di Vito. Formalizing space shuttle software requirements: Four case studies. *ACM Transactions on Software Engineering and Methodology*, 7(3):296–332, 1998.

[6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1998.

[7] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium 1993. Proceedings*, pages 141–152. The Institute of Electrical and Electronics Engineers, 1993.

[8] H.S. Delugach. Specifying multiple-viewed software requirements with conceptual graphs. *Journal of Systems and Software*, 19(3):207–224, 1992.

[9] M. Dorfman. Requirements engineering. *Software Engineering Institute Interactive*, 3:1–30, 1999.

[10] S. Faraj and L. Sproull. Coordinating expertise in software development teams. *Management Science*, 46(12):1554–1568, 2000.

[11] T. Gorschek and M. Svahnberg. *Engineering and Managing Software Requirements*, chapter 18. Requirements Experience in Practice: Studies of Six Companies, pages 405–426. Springer-Verlag Berlin, 2005.

[12] Saab Group. Electronic defence systems. Accessed: 2010-05-17, 13:09, `http://www.saabgroup.com/en/About-Saab/Company-profile/Organisation/Electronic-Defence-Systems/`.

[13] Saab Group. Saab in brief. Accessed: 2010-05-17, 12:59, `http://www.saabgroup.com/en/About-Saab/Company-profile/Saab-in-brief/`.

[14] The Standish Group. Chaos: The standish group report. Accessed: 2010-03-11, 12:01, `http://www.projectsmart.co.uk/docs/chaos-report.pdf`, 1995.

[15] G. Kotonya and I. Sommerville. *Requirements Engineering*. Wiley Chichester, 1998.

[16] S. Lauesen. *Software Requirements: Styles and Techniques*. Addison-Wesley Professional, 2002.

[17] K. Lyytinen and D. Robey. Learning failure in information systems development. *Information Systems Journal*, 9(2):85–101, 1999.

[18] The Institute of Electrical and Electronics Engineers. Ieee standard glossary of software engineering terminology, 1990. Std 610.12-1990.

[19] M.C. Paulk, MB. B. Curtis, Chrissis, and C.V. Weber. Capability maturity model for software, version 1.1. Technical report, Software Engineering Institute, 1993.

[20] B. Regnell, M. Höst, J. Natt och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, 2001.

[21] U. Reja, K.L. Manfreda, V. Hlebec, and V. Vehovar. Open-ended vs. close-ended questions in web questionnaires. *Advances in Methodology and Statistics (Metodološki zvezki)*, 19:159–177, 2003.

[22] W. Robinson. Monitoring software requirements using instrumented code. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 276–276. Citeseer, 2002.

[23] W.W. Royce. Managing the development of large software systems. In *Proceedings of IEEE Wescon*, volume 26, pages 328–338, 1970. Accessed: 2010-03-03, 10:52, `http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf`.

[24] A. Ryen. *Kvalitativ intervju - från vetenskapsteori till fältstudier*. Liber, 2004.

[25] I. Sommerville. *Software Engineering*. Pearson Education Limited, 8th edition, 2007.

[26] Statistics Sweden. Swedish electronics industry and companies in the ict service sector 2006-2007. 2009. Accessed: 2010-03-07, 11:33, `http://www.tillvaxtanalys.se/tua/export/sv/filer/statistik/Statistik_2009_06.pdf`.

[27] W.M.K. Trochim. The research methods knowledge base. Accessed: 2010-05-06, 09:34, `http://www.socialresearchmethods.net`.

[28] K. E. Wiegers. *Software Requirements*. Microsoft, 2:nd edition, 2003.

APPENDIX

A. KEYWORDS

The following keywords where used to search for previous research in the software requirement area:

- software division
- software requirement
- software requirements
- software validation

B. GLOSSARY

This is a glossary of the abbreviation used in this paper.

- Saab EDS = Saab AB, Electronic Defence Systems, Operations Göteborg

C. INTERVIEW QUESTIONS

The below interview questions were used during the five interviews. Here they are translated from Swedish to English, by the author of this paper.

General Questions

1. In what product area do you work?
2. Do you have a division of requirements in different levels? (If yes continue to question three.)
    a. If no: Why do you not have that?
        i. Would you like to have a division of requirements?
        ii. Terminate the interview.
3. How does the division situation look like concerning software requirements, for example do you sit in a division meeting or do you do the division by yourself? (If they are by themselves go to question five.)
4. What does the discussion concerning software requirements look like? (What argumentation do people to get their point across?)
    a. Does the discussions look like you want them to?
        i: If yes: Why?
        ii: If no: Why not?
5. How would you want the software division process to look like?
    a. If another way: How would you want the software division process to look like?
6. Is it something characteristical in the software requirements themselves that determines the division?
7. Does the practices software requirements division models look like any of these models? (Show Model paper.) You also have the option to draw your own model. (Show blank paper and pen.)
    a. If one of the software requirements division models but not exactly: What differs?
    b. If many of them: Is it something in the software requirement or something else that determines the model used?
8. Are their software requirements and division of software requirements that always are present in every project or product?

Documentation

9. How is the division of software requirements documented? (Are there many documents based on different level and do they look the same?)
10. Do you want the division to be documented in that way?
    a. If yes: Why?
    b. If no: How do you want it to be documented?
        i: Why?
11. Who is the recipient of the software requirements division document and do you think it effects the division of software requirements?
12. Do you see that the documentation of the software requirements division is necessary for you to perform your work?
13. Are there aspects of the software requirements division that you think can be removed?
14. Are there aspects of the software requirements division that you think must be kept?
15. What do you see can be the effect of not documenting the software requirements on the right level?
16. If there are many documentation places: What can you see as the effect of not documenting the software requirements division in the right documentation place (for example in the right document)?

Validation

17. Is the validation effected of how the software requirements are divided?
    a. If yes: How?
        i: Is it always like that?
    b. If no: Why not?
18. How do you want the division of software requirement in the aspect of validation to look like?
    a. If yes: Why?
    b. If no: Why not?
        i. How would you like it to look like?
19. Is there information about how the software requirement are divided that need to be present for the software requirement to be able to be validated?

Verification

20. Is the verification effected of how the software requirements are divided?
    a. If yes: How?
        i: Is it always like that?
    b. If no: Why not?
21. How do you want the division of software requirement in the aspect of verification to look like?
    a. If yes: Why?
    b. If no: Why not?
        i. How would you like it to look like?
22. Is there information about how the software requirement is divided that need to be present for the software requirement to be able to be verified?

End Questions

23. What can you see as the effect of not dividing software requirement on the right level?
24. Would you like that documented information about

how the system should function was presented in another way? (In other words not in by the book requirements but in for example exploratory development, models or so on?)

25. Is there something you would like to add?

D. DIVISION MODELS

The division models on this page were used during the interviews. Here the are translated from Swedish to English, by the author of this paper.

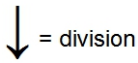

Figure 2: Model Description

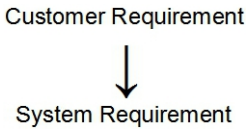

Figure 3: Model 1 - Software Requirements Division

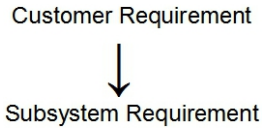

Figure 4: Model 2 - Software Requirements Division

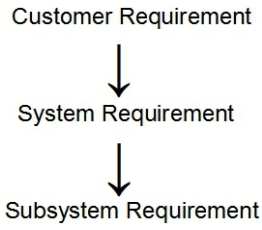

Figure 5: Model 3 - Software Requirements Division

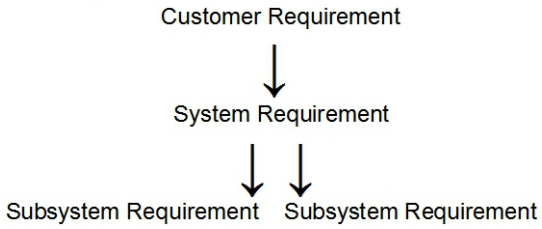

Figure 6: Model 4 - Software Requirements Division

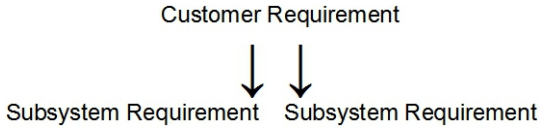

Figure 7: Model 5 - Software Requirements Division

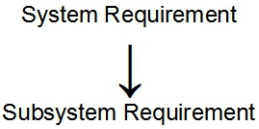

Figure 8: Model 6 - Software Requirements Division