LUNDS UNIVERSITET
Lunds Tekniska Högskola

Master Thesis

Managing and utilizing dependencies between components in component-based systems

Authors:
Tobias Landelius, tlandelius@gmail.com
Viktor Attoff, viktor.attoff@gmail.com
Supervisor: Lars Bendix

Lund Institute of Technology
Lund, Sweden

August 2017

Abstract

When developing software systems the advantages in using a modular architecture are many, e.g. scale-ability, re-usability and the ease of making changes. However, this type of architecture creates problems regarding how modules depend on each other when doing change impact analysis, communicating about the system and creating awareness of how a system works to name a few. Today many kinds of dependencies are not documented and left out of the development process. With no generalized structure or definition of dependencies, developers might overlook problems that could be found proactively.

By literature research and qualitative interviews of industry people in various roles, this thesis elicits the core problems around dependencies. Managing dependencies is seen as complex, and dependency related problems can have a severe impact on development if caught late. The thesis presents that an overview-oriented dependency management process can benefit software development regarding estimating change-impacts and cost-of-change, aiding communication during development and optimizing the amounts of tests that have to be run after a change. The results present the core problems of dependency management, use-case around them and theoretical requirements of how dependency management can be conducted to deal with the problems.

Keywords: Dependency management, Dependency utilization, Change control, System awareness, Impact analysis, Communicating changes, Test-optimization

Acknowledgments

First of all, we would like to offer our grateful thanks to all those who gave their time and expertise to help us during this thesis.

A special thanks to our supervisor and mentor, Lars Bendix who has shown a great interest in this thesis from the start. He has been a valuable source of inspiration and feedback when writing the report.

We would also thank Softhouse Consulting and our company supervisor Fredrik Stål with colleague Christian Pendleton for bringing us this Master's Thesis and being a great sounding board on topics from finding people to interview to help us understand the field of dependencies.

Finally, we would like to thank the SNESCM community and interviewees that took their time helping us to reach the results of this thesis. It would not have been possible without you.

Contents

List of Figures

List of Tables

1 Introduction

As software systems grow larger, the need for an efficient development process becomes crucial for keeping the progress from stagnating. Changes and improvements are regularly made on many different levels to streamline the process and minimize obstacles. One such area of change is the change from a monolithic system into one with a modular architecture. By building systems with as much modularity as possible, it's possible to avoid tightly coupled code and leads to several development advantages, for example, better scalability, reusability and ease of making changes. Although a system with modular architecture is divided into separate modules, they have to coordinate with each other for the entire system to work. Therefore we want to make it possible to create a set of inter-modular dependencies within our system.

This master thesis was done in collaboration with Softhouse Consulting Öresund AB. They brought the suggestion of a master thesis that would investigate and explore the subject of inter-modular dependencies. Which was based on that during their consultancy at different companies they recognized different reoccurring dependency-related issues.

1.1 Problem Description

As modular systems are becoming more frequently used and larger, dependencies between modules are becoming a more significant factor in development. As dependencies describe how modules communicate, they occur in many different types throughout a project. Some already documented, some indirectly documented and some never even considered as a dependency. By leaving out the dependencies from being a direct factor taken into consideration in the development process, developers might miss out on opportunities. Using the dependencies could help teams work proactively against inconsistencies and problems between modules rather than only reacting to problems as they occur later in the development process, or even after deployment to a customer. One additional problem that follows with dependency management, as well as most documents that belong to a project, is how to keep the documentation up-to-date with the system.

The aspect of missing documentation of dependencies may not lead to a broken system, but as important it limits the communication in the development process. The lack of management of dependencies has revolved in no standardized way of referring to a particular relation between two components as the code that creates the dependency may change continuously with the system.

1.2 Research Questions

Problems like the ones mentioned above ignited the idea that they might be a possibility to solve by generalizing dependencies and define a process of how to utilize them. After some initial research and discussion of how to approach the subject of dependency management, two research questions were extracted.

First is to find the core dependency related problems that could be solved with the help of dependency management. Also giving a complete view of *Which problems occur most frequently?* and *Which problems cause the biggest issues to the developing project?*. Finding these problems is important to get a sense of prioritization and what to include in the reasonable scope for this thesis.

> **Research Question 1:** What problems do we want to solve by documenting and managing dependencies between components?

Once the core issues were elicited, the next step is to find a way to deal with them. To be able to find a good way of doing this there is need of knowledge of the structure of a dependency and what characterizes it. Questions like *What specifies different dependencies? How can dependencies be identified? Once identified, how can they be updated along with the project?* came forth and these were some of the questions that needed to be considered to start moving towards a solution to the problems that were the outcome of research question 1.

> **Research Question 2:** To solve the problems of risen in RQ1, how can dependencies be documented and utilized?

Research question 2 will result in requirements for a suggested general process for managing dependencies. Therefore there is a need to return to the to a practical setting to validate the results. Validation will mainly revolve around if companies believe that a full or partial solution based on our suggestion could be useful in a practical setting.

To answer these questions, we will start with information gathering by both literature search and interviews with multiple companies that use component-based structures in different ways. By analyzing the information we'll be able to extract requirement of how dependency management can be performed.

2 Background

The goal of this chapter is to create a wider understanding of the context for this thesis. To fully proclaim why this area is important, this chapter will start out with a further motivation of the research questions by describing initial problems that can be related to dependencies. Since the definition of what a component is and what's included into a Component-based Structure varies a great deal, a formal definition will be presented as this thesis firsthand researches the area of dependencies between components. A basic understanding of Configuration Management (CM) is profitable for the reader to fully understand the area to be explored. Therefore a short introduction will be made to some of the main principles that will be adopted in this thesis and the context of the processes for targeted projects. In the final section of this chapter, it will be presented to whom this thesis is written and what projects are targeted.

2.1 Problem motivation

This thesis approach to the problem originated from some specifically encountered real life problem that was brought forwards as a master thesis idea by Softhouse Consulting. These problems combined with a vague idea that there is much potential in the area of dependency management in modular architectures where the only starting point. Softhouse also initiated the idea that the types of problems that they encounter were by nature mostly dealt with ad hoc. If there were a way to incorporate dependencies into the development process and manage them, it might be possible to do proactively and become aware of a potential upcoming problem much earlier in the process. The vague starting point motivated the need to elicit further what kind of problems are the most costly, appear most frequently and the some of the consequences they bring, which lead to research question 11.2. There was seven interview in 4 companies involved in this elicitation process. But making sure the interviewees have different roles within the companies combined with thorough literature search hopefully motivates the suggested solutions to be either generalized or at least inspirational for any software development project.

Dependencies is something that has always been a part of software development. Early definitions were presented in the early 70s, a popular and well-cited one being from Stevens et al (1974)[24]:

> *"A dependency is the degree to which each component relies on each one of the other components in the software system. The fewer and simpler the connections between components, the easier it is to understand each component without reference to other components."*

Since then, further work has been done involving dependencies but not enough to keep up with the evolution of software development. There are currently several different methods for to handle dependencies at specific areas [7], but research also shows that there is much potential to develop new methods[19] [14]. The subject of this thesis was suggested by Softhouse Consulting that encountered problems in software development projects that in different ways were related to dependencies. This thesis aims to further look into that potential of managing dependencies and specifically in the context of modular architectures.

The increase in popularity and scale of using modular architecture is one motivation to further research the possibilities of managing dependencies. As a modular design means breaking up a system into sub-parts, this makes the dependencies between them more visible and specific than in a monolithic system. The same dependencies would exist if the system were monolithic. But they would be in an embedded in the monolith, and their existence and purpose would be less visible. One of the main reasons to choose a modular architecture is to have a system design that is more scalable. With the ability to more easily create larger systems the task of maintaining an overview of the entire system becomes more complex. To keep an overview not only components are important, but also their dependencies. Furthermore, in larger modular systems, responsibility for modules are often divided among teams. Which means that a team responsible for some modules doesn't necessarily need to have great knowledge about other modules. For the team to develop the intended system, there is a need to be aware of how their modules communicate, interacts and fits into the greater system.

Another initial motivation for this thesis was the fact that there are dependencies related to modules at different levels in development and perhaps there are missed opportunities by not using this information in the development. These different dependencies also vary in complexity. Simple dependencies like those listed in a Gradle-file show dependencies to third-party modules and libraries we need to build our system. A module that fetches data from a database has a dependency between them, which could be indirectly documented through requirements. Another more complex possibility is that dependencies might be documented on an architectural level with the help of UML-diagrams or design sketches. These are all examples that somehow involve dependencies at different levels in a system. Looking at different dependencies like these and bringing them into the working process could increase the ability to work proactively against problems instead of having to react when they occur late in the process. An initial view for this thesis was that only reacting to problems around dependencies is a common occurrence and brings uncertainty to the process, which motivates further research.

2.2 Component-based Structure

There are various definitions of the term, Component-based Structure (CBS) as well as different scopes of what is meant by a component [25]. The paper by Wu, Pan and Chen provides a definition that is often used, that *"a component is an independently deliverable piece of functionality providing access to a service through interfaces"*. This definition is most often thought of from a software perspective and that the functionality is provided as a piece of code that is here defined as a *module*. For this thesis, it's not only the software aspect that is thought of when easing the problems that arise when developing with a CBS. When software is executed, it might depend on components in the surrounding environment.

In a lot of the projects that utilize CBS's, the dependencies do not have to be between two software components; they might be directed to the presence or limitation of a hardware component. This might take place as a feature is in need of a Bluetooth-device or a specific amount of memory to work properly. Because of these dependencies, the aspect of hardware devices is included in the definition of components and might be a part of a CBS.

2.3 Configuration Management

Configuration Management is the domain in software development which this thesis is present. According to an investigation of configuration management usage by Perera et al., benefits that can come with the integration of the CM process include minimizing the cost of calculating impacts of change, coordinate, track and manage change activities as well as an improved breakdown prevention due to a more controlled development environment [20]. An observation is that there is a large amount of research that supports that these areas correspond to the ones that managing dependencies can benefit [21] [8] [17]. A large majority of all companies that evolve software use configuration management to some extent, Perera discovered that 95% of the investigated companies used CM for the development of software..

There are two major sub-process in CM that can benefit from the findings that will be explored. Pre-change analysis and testing. In many agile development processes, especially the pre-change analysis can be small in comparison to the more traditional waterfall methodology [2]. As the complexity of the dependencies scales with the project, even agile companies might benefit by introducing a more substantial pre-change analysis if these problems exist, as Lopez-Martinez et al. present the risen difficulty in keeping a fully agile methodology in large-scale projects [18].

To be able to fully profit from the outcome of this thesis a project need version control to store and handle code and dependencies. Version control can be achieved with the help of several different available tools. The activities in CM that this thesis contributes to is configuration identification which derives from version control. Configuration identification, in this case, includes the subject of identifying new configuration items that are introduced to the project in the form of dependencies, naming and creating versions of these that can be connected to existing baselined versions of the system. These dependencies can then be utilized to aid in many of the subjects revolving configuration control and the planning process, including change documentation, impact analysis, to calculate the cost-of-change and lowering the risk of introducing faults into the system.

2.4 Target Audience

The target audience for this thesis can be divided into three groups. Firstly it would be interesting to people in any role related to software development that is interested in managing dependencies. It should be in any developers interest to want to further improve their working process. It is especially interested to people working in medium to large size projects was dependencies should play a larger role than in small projects. The second target audience is people who in some way are interested to continue the work in dependency management. This could be other student or researchers looking for material for future research. It could also be one or several people looking to make a practical tool implementation in this area. Lastly, since this thesis is aimed to tackle dependency management from a software configuration management perspective part of the target audience is also people with a general interest in SCM

Common for all target audience groups is that they have some knowledge of software development and its different processes, either from university courses or work experience. This means some processes and expression that are referred to in this thesis will not be explained and is up to the reader to further study if necessary.

3 Methodology

As the research questions are quite general, there were several different approaches to take on these questions. This chapter will present the methodology that will be carried out for this thesis. The methodology is chosen with consideration to what is the most optimal, practical and possible in the context of a master's thesis. Therefore this chapter starts with a methodology overview of the taken steps, followed by each step where it also discusses alternative methodologies that wasn't chosen and why this was the case.

3.1 Thesis work methodology

As the thesis begins, the expectations is to gain knowledge on dependency management can be useful in a company context. To reach this outcome it's first off recognized a need for a deeper understanding of the actual problems and context of processes around dependencies within a CBS. Therefore the methodology will start out with a information gathering process. The information gathering starts with the purpose of widening the scope to explore what's included into the area of dependency analysis and management. As the scope is widen and a lot of information is gathered there is a need for an analysis. During the analysis, data is structured and evaluated. This analysis will then work as a basis during for the results of this thesis. Exactly how the results will be presented is dependent on the findings and evaluations from the previous steps. When an result has been produced the question remains how well this could fit into the companies that utilize CBS's in their development. The thesis is finalized with a validation where the explored results is taken back to the context of the companies to see to what degree it could be useful as a process or as a foundation for future work.

There is an uncertainty in how large and complex the existing problems are. As the context and the scope of the problems becomes clear, it will decided if a tool will be possible to carry out in the given time frame. If the scope becomes to large it might not be able to create a tool that satisfies the existing problems in the companies. If this is the case, the priority will be transferred to a analytic standpoint and work with the given information and focus the research into selected parts of the scope. The theoretical approach would focus on creating knowledge and insight of the problems, why they exist and how they could be managed and used. Leaving the tool as future work.

3.2 Information gathering

As the thesis progress, there's a lot of needed information to create a suitable ground of research to continue working on. To acquire a knowledge of where to begin there is a need of related work and a status on what research has been done before this thesis. Due to the difficulty of finding a broad amount of information on a relatively unexplored subject, this is done with the help of literary research.

Secondly, there's a need for some information regarding the usage and processes related to projects built upon a CBS. To acquire this information, there are three main methodology methods risen that can be used, interviews, questionnaires, and observation. Due to the limited amount of time and uncertainty of finding valuable information, observation of development in companies was quickly removed as an alternative. The advantage of using interviews over questioners is the ability to explore more thoroughly into interesting findings with the help of follow-up questions and if an answer is not understood, the possibility of asking for an explanation. This results in multiple interviews with people from different size companies that all have a different relation to developing software systems.

3.2.1 Literary Research

The literature research was performed on several occasions during the creation of this thesis. For each of these occasions the research was performed in an iterative cycle of three steps that can be seen in figure 3.1. There is four phases during this thesis where there's a need for extracting information, these phases are pre-analysis, primary literature search, paper review and extensive paper review. To make sure that each phase add new information, they have a specific objective of what it will provide in terms of material and what search engine that will be the main source.

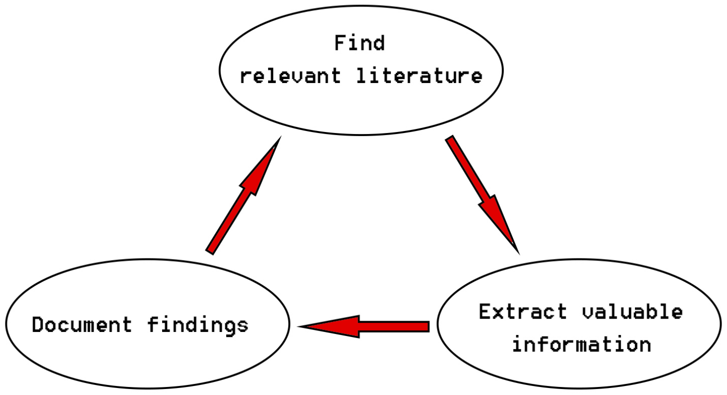

Figure 3.1: The iterative cycle of the steps in literature search.

The process starts by finding interesting literature. To find a wide amount of different literature, this is done with the help of a search engine and a combination of m. The set of keywords that are used will be created by saving keywords that provide relevant literature that concur with the objective of the phase. When the information gathering process starts off with a wide scope, the keywords will follow the same pattern. As the scope is directed into relevant areas, so will the keywords as they will be prioritized and refined.

The research started with a pre-analysis of the area as an introduction to the existing utilization and problems regarding dependencies. Google's search engine was the primary source to find a large amount of literature from, e.g., blogs, articles, and YouTube-videos. If the information was relevant to the thesis, the source was documented as a possible reference for further research. The purpose of the pre-analysis is to find a large amount of easily digestible information in the field of CBS's and dependencies along with some of the more well-known problems in these areas. This is needed for two reasons. First, it's possible to narrow a large scope down to the areas related to the research questions and by being aware of the areas makes it possible to create a set of interview questions to investigate where a company's potential problems lie.

When the pre-analysis is done, and an initial knowledge in the subject has been acquired, a primary literature search is done. For the primary literature search, the source is transferred over to Google Scholar and LUBsearch to find a more reliable range of articles, journals and conference papers. A preliminary evaluation of the relevancy of paper will be done by reading the abstract and graded from 1 (*not so relevant*) to 3 (*very relevant*). Except only using keywords to search for new literature, Google Scholar has a built-in functionality called *'cited-by'* that may be used when a good reference is found [15]. The purpose of doing this iteration was to create a suitable amount of literature to perform the first paper review on.

The first paper review is performed on the papers that are graded 2 and three from the primary literature search. The papers are read, and all relevant information, citations, and claims are marked and documented. After the information extraction from each paper, a short bullet list of findings is created as the grade of relevancy is updated. The purpose of doing this is to create a set of papers that all are relevant to the research area of this thesis and the summary is created so the correct paper easily can be found.

The last process of the literary research is an extensive paper review. Due to time limitations, four relevant research areas are chosen, dependency utilization areas, dependency structure, presentation of dependencies and the communicative challenges arising from disregarding dependencies. The most relevant paper within each area is chosen for an extensive paper review according to Philip Fong's structure [12]. The purpose of doing this review is to correlate the fundamental parts of this thesis to the most related work that was found during the literary search.

3.2.2 The interview process

To acquire a first hand understanding of the problems and countermeasures revolving dependencies that arise in a company context, interviews will be performed. To collect a wide amount of different perspectives, three different companies will be selected for the interview process that all have different development strategies to CBS systems. The first company is a large software distributor with an agile approach and relatively short time between deployment, second is a large company that develops a safety-critical software that deploys annually. Due to the safety-critical software, the development methodology is strict, and change is highly processed before implemented. The third company is a consultant company that has been in various development surroundings that include CBS. There were nine interviewees with different roles in companies; three project managers, one configuration managers, two architects, and three developers. All having various focus but all interested in configuration management and the subject of this thesis. This is once again to widen some perspectives to CBS's.

To make sure that all interviews add qualitative data, the interview is conducted in a semi-structured manner. Some questions are pre-chosen that touches different areas of development processes, dependencies and configuration management. These questions are considered important as they give an overlying structure of the interview and to make sure that some central questions are answered by every interviewee. These questions with the motivation of the wanted outcome can be found in Appendix A. By doing a semi-structured interview, it opens the possibility of further investigating findings or following up on answers that are considered interesting.

3.3 Analysis

The overall goal of the analysis in this thesis is to analyze the information gathered through literature and interviews and finally arrive at the results. Which was initially meant to be a proof-of-concept tool but later changed to requirements for a tool. The pre-analysis showed that the subject of handling of dependencies seems rather unexplored which is why the interview and information gathering was done with an exploratory approach. This lead to the varied type of answers and focus area depending on what company the interview took place and the title and expertise of the interviewee. This lead to interesting, in-depth answers but difficult to make some statistical analysis on or compare to each other. Therefore a qualitative analysis method was done to extract common core factors that needed to be taken into account to answer the research questions. The steps of this analysis consists of problem statement, use-cases and requirement is described and motivated in detail in section 3.3.1 to 3.3.3 below. The overall goal and purpose of this analysis setup are to focus the information gathered into its core values and then work towards a suggested solution. Problem statements are then created to narrow down the information gathered from interviews to a set of representative problems. Then to further explore and understand these problems use-cases were created. Finally to combine this knowledge with previous research from literature and new ideas to form the results in the form of requirements.

As mentioned in section 3.1, halfway through analysis of previous research and interview results the real scope and complexity of the problem became apparent. Early on the intention was to produce a result in the form of a simple tool or a proof-of-concept that could demonstrate the problems in this domain and how to facilitate with them. But because of the complexity and time constraints, this proof-of-concept would turn out very basic, and making it would take time away from a complete theoretical understanding and information processing. Therefore the target for final result was changed to a requirements specification for a possible future tool. This was a more appropriate scope and lead to maximum possible contribution for this thesis.

3.3.1 Problem Statements

The first step in the analysis of material is to narrow down the information from the interviews to problems statements that were representative of the most common and are the most common. The alternative would be to use every problem found in the interviews and brought them into the consideration for the requirements design. This would be too complex to begin to design requirements around and difficult to satisfy every problem. Since this thesis is an initial examination of the problem of dependency management, it is better to only the most worthwhile problems. Also in this stage of the analysis problems are also analyzed regarding if there is a realistic chance to address them. Some problems gathered from the interviews were directly at the interviews or shortly after disregarded and agreed upon as unrealistic and too complex. In this step, no information of possible problems is matched with existing literature. This step is done with the purpose of focus on the problems faced at the interviewed that should be continued to further analysis.

3.3.2 Creation of Use-cases

Once these problem statements are listed there is still a need to further explore the problems. This is necessary to get more understanding of how these problems occur in a practical setting. Use-cases help discover in what context the problems will arise, what types of projects, in which stage of the development process they occur, what roles are involved and what the implications are. The main reasons for the use-cases are to build this understanding before extracting requirements in the next step of the analysis. But the use-cases could still be seen as secondary results of this thesis because they are a product of the research and provide value to anyone that wants further insight into this way of managing dependencies.

3.3.3 Requirement extraction

The final step of the analysis is to combine above mentioned intermediate results into a presentable suggested solution. As mentioned and motivated previously it was decided to disregard making a proof-of-concept tool and instead present the result as requirements for a tool. Mainly because of the scope and unexplored nature of the problem presenting requirements will allow for a more in-depth exploration of the problem. The requirements are forward as a combination of all previous work. A lot of ideas developed continuously throughout the research, many of which were disregarded as the understanding of the area grew. The good ideas were then combined with the problem statement, use-cases, and the most related literature to form the requirements.

It was pointed even before the thesis work started by all parties that this is a complex, unexplored problem and it will probably be unrealistic to create a fully complete requirement specification. Therefore focus lies on putting in the most effort and focus on those areas of which feasible ideas and solution are produced. Leaving out some areas for future work rather than presenting incomplete ideas in the requirements. To make sure that every problem is taken into consideration when creating the requirements the process of extracting requirements is described by the flow chart in figure 3.2.

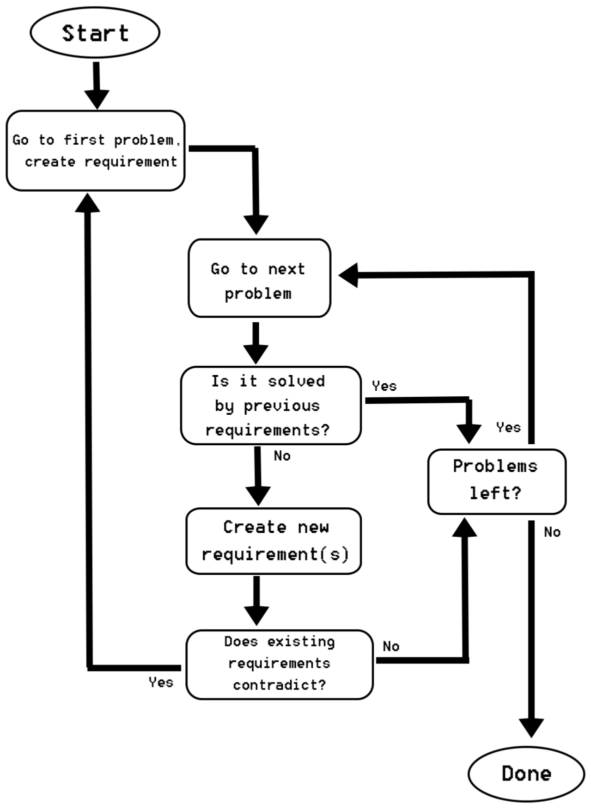
Start
Go to first problem, create requirement
Go to next problem
Is it solved by previous requirements?
Yes
No
Create new requirement(s)
Does existing requirements contradict?
Yes
No
Problems left?
Yes
No
Done

Figure 3.2: This is the process for high-level requirement extraction which purpose is to make sure that every category is considered.

3.4 Validation

Once the results are in place there is need to validate them with a practical setting in mind. This is important because theoretical work often assumes simple, ideal conditions as their context but that is seldom how projects in the industry look like. Two different methods of validation are considered. Either a quantitative method with a questionnaire about the requirement design sent out to relevant people in the industry. This will allow for a large sample of answers and the possibility of statistical analysis. The other method is the qualitative method to return to the interview setting. This will result in more in-depth back and forth discussion about the requirement design which means more ideas and input.

Out of these options validation interviews seems like the most suitable for validating the requirements design. The nature of the requirements makes it hard to collect enough in-depth validation with a questionnaire. Before people can answer a questionnaire a lot of material would need to be presented in text. This would bring the risk of lowering the number of people willing to complete the questionnaire. A major factor that speaks for validation by interviews is the possibility to ask appropriate follow-up questions of why the interviewee have the opinions that they do.

Apart from qualitative interviews, there is also an opportunity to hold an open-space session/workshop at a configuration management conference held by the SNESCM-organization. This will be done about two thirds through the thesis is complete and will give an opportunity to get input on ideas and initial findings and results. Hopefully, it is possible to critique on what people in the industry think about the findings and if they deem them to be reasonable to work in a practical setting.

4 Analysis

This chapter will present the analysis done on the information gathered from literature and interviews. It will describe the outcome of problem statements and use-cases with some alternative considerations. This will be presented as the three step process, create problem statements, use-cases and requirement extraction described in section 3.3. The primary purpose of the creations of problem statements and the use-cases is to create a ground that leads up to requirements. These can still be seen as a contribution to the understanding the overall problems of dependency management and especially to the first research question. Therefore the secondary purpose of this chapter is to present secondary results.

4.1 Problem Statements

The information gathered from the interviews include a wide spectrum of problems both in and outside of the scope of managing dependencies. From the problems extracted from the interviews, after being discussed and compared, some problem statements are selected as potential items to Research Question 1 (*What problems do we want to solve by documenting and managing dependencies between components?*) stated in the introduction. This section will present the selected problem statements that were extracted from the interview data. There are some problems that multiple interviews brought forward that was selected as out of scope for this thesis, discussion around these can be found at the end of this section. The statements are listed in order of first, how the interviewees look at the problems regarding severeness and a final judgment by the authors in how relevant the problem is to the scope, giving the most relevant problems first. This is to provide a priority of dealing with the problems.

Problem 1: Change impact analysis

- Hard to see an impact of a change and find the root-cause of a problem.

Working with a CBS helps to isolate functionality and purpose into specific modules. A difficulty that arises is that if something breaks after a change, the root-cause of the error may be located in another dependent module rather than the changed module itself. Tracing the root-cause and calculating the impact of a change is a hard task and is often done manually by the companies as the relationships between components are overlooked.

Problem 2: Awareness

- It is hard to gain insight and overview of a system.

Several companies described it as a difficulty to get an overview of the entire system or a part of it. This was especially expressed in interviews with architects, who claimed it's common that new developers that get assigned to work on one module often have a hard time to understand the modules purpose on a larger scale and the modules that it related to.

Problem 3: Communicate change

- After a change is made it's hard to know what and to whom information needs to be communicated.

When a behavioral change is made is a system, two companies used the process of communicating the change through email. Either the email went out to the entire company to make sure that everyone was up-to-date of the change or to a grouping of members that was somehow related to the change. Either way, a number of emails and the information in them became too large in these two large companies and it was rarely worthwhile taking the time to read every message.

Problem 4: Build requirement

- Not being aware of what is needed to build and use a certain module.

This problem arose in one of the interviews as the projects consisted of modules that were worked on or tested by different teams. When only interested in the functionality of a certain module a team still had to collect the modules for the entire system as there was no way of knowing the minimum set of required modules needed for the desired module to run. This lead to substantial time spent on sending components that might be unnecessary for or already present in the receiving team.

Problem 5: Test optimization

- Not knowing what components need testing after a change is made.

After a change is made to a module it might affect other modules. Two of the companies had problems with long testing times due to having to test the entire system as a whole after each change. This becomes an optimization problem as it does not break anything but the process that is used, is not optimal.

Problem 6: Alternative targets

- The same system might utilize different modules depending on its environment.

Two of the companies had examples of problems regarding a system that is developed to fit for multiple hardware targets. In this system, it exists multiple software components where each one is dependent on a specific characteristic of the target where it's installed e.g. the amount of memory in the device. This is often handled in the code but this alternative dependency of components is often lost in the overview of how the system works.

Problem 7: Corner case study

- Finding and being more aware of corner cases of the program.

All companies had experienced some errors that occur in a later stage of development or even after deployment. These errors are not caught during testing and there is no knowledge of the error's existence. The definition of the term *case* can be found in the IEEE glossary [3] as *corner* refers to a situation outside of the normal circumstances which make them hard to find and produce. This is a big problem in most projects but due to the difficulty of finding all circumstances that are out of the ordinary, this is given a low priority.

The primary purpose of creating problem statement was to do an initial screening of all problems discovered during the interviews and filter out problems that are too context specific, not strongly enough related to dependencies or for other reasons that are considered as out of scope for this thesis. The following two paragraphs will shortly mention the most interesting out of the left out problems and why they are left out of consideration.

An issue that was suggested during interviews is that companies that maintain multiple large projects spend too much time on creating modules with duplicated or overlapping functionality as modules in a different project. The thought being that if two completely different projects had a better awareness of what other teams are developing, it would be possible to develop the intended functionality as a shared module instead. This problem was deemed closer related to functionality and requirement management than dependency management and therefore left out of scope. In the scope of requirement management it was also elicited that one or multiple components could have a dependency to the presence of a functionality, e.g. a module where a zoom functionality (for a camera) is implemented, is depended on the presence of the functionality of a camera to work as intended. This problem is related a functionality and requirement management as well and is therefore acknowledged as out of scope for this thesis.

All companies and most of the interviewees had problems with the identification of dependencies, in other words, *finding* the dependencies especially within an existing system. During the literature research, it was found plenty of research made in the area of automatic dependency identification (more on this area in the related work section, 6.3). From both the literary findings and information extracted from the interviews it is found that this area is a hard and time-consuming task. With the risk of taking too much time trying to solve this problem and leave too little time for other problems, this was removed from the scope of this thesis.

4.2 Analysis of Use-cases

The way the problems above are stated it's not yet clear how or why the problems arise, only that they exist. To put the problem in a context and to help to figure out some of the main properties to what kind of projects and environments it might occur in, use-cases are created [9]. As the last step of the analysis is to extract requirements from the use-cases, it's not only a need of a better insight of the context of the problems but also an idea of how the problems may be dealt with. This idea will be used in the next step as a base to extract the requirements. This following section will provide analysis on how dependency management is integrated into the problem statements and a use-case to put each problem into a context as it might present itself. This use-case is then further analyzed to create a general concept to how dependency management could be utilized to solve the problem. A more detailed version will be created later in the form of requirements.

Each of the following subsections will present the analysis and use-cases relating to one or two problems (P) from the previous section. To create a context to the problem statements the sections will start with a motivation of why the problems are categorized as they are, followed by a further analysis of the actual problems and some of the difficulties that may arise when dealing with them. For each problem, a use-case (UC) will be created and presented to create a better understanding of how the problems might occur in the context of a company. If a problem presented itself in two very different contexts during the interviews, an additional use-case will be created to obtain a complete view of the problem. Each use-case consists of a short description of an event that is based on the analysis of the problem and how it could present itself in a company. To broaden the understanding of the context, a scope is presented in each use-case to explicate to whom and where in the development process this occurs. As the use-cases are created from the problem statement, an overview of this relationship can be found

	P1	P2	P3	P4	P5	P6	P7
UC1	x						
UC2	x						
UC3			x				
UC4		x					
UC5		x					
UC6							x
UC7				x			
UC8					x		
UC9						x	

Table 4.1: Problem and Use-case relations

in table 4.1 as a cell with a marking 'x'.

4.2.1 P1: Change impact and P3: Communicating Changes

The first category of problem statements includes some of the most highly prioritized problems extracted from the interviews, change impact and communicating changes. By analyzing these two problems it's found a shared way of how they can be dealt with. Since root-causes of both problems include lack of knowledge in which and how modules affect each other by changes.

Problem 1 is about the difficulty of doing a correct change impact analysis of a system. This analysis is something that either can be done beforehand to take care of errors proactively or in the process of finding root-causes to errors that already occurred. The process that is done before a change, including i.e., change impact analysis and calculating the cost-of-change is often performed in the presence of a change request. These processes take a substantial amount of time and if they're done incorrectly, it can lead to expensive errors discovered much later in the development process. If an error already exists in a system it can be a tedious and time-consuming task to find the root-cause to the problem. This problem becomes harder to manage and more complex as the system is divided into components without good knowledge of how they relate to each other. These two contexts of the same problem result in two use-cases.

UC 1: Pre-change analysis of a system

Event: New requirements lead to major changes in a system which creates multiple change requests that need to be processed correctly.

Scope: Before a change is to be made to the system that requires pre-change analysis. Specialists, project leaders, architects and configuration managers are some of the roles that are involved in these processes.

UC 2: Analyzing the impact of a change

Event: A change has been made to several components of a system. When running the system, the execution fails in a component that wasn't changed.

Scope: During development and maintenance a lot of time is spent by architects, developers and testers trying to achieve a correct change impact analysis.

Problem statement 3 relates to the hard task of communication. This problem was also highly prioritized among the interviewees as it is very challenging to provide the correct information about a change to a correct set of people. Many companies use the good conduct of communicating their changes throughout the project, in large projects these notifications aren't always read as the amount becomes overwhelming due to a lot of changes. The problem shows itself in two questions that are hard to answer, first being the question of *who's the involved parties that need to be notified?* and the second being *what information needs to be notified?*. The first question is related to dependencies and finding modules that relate to the one being changed. The answer to the second questions differs slightly depending on if the related module is from an internal or external party. An internal party often need short and informative information about the change and why it's made while an external party often need more extensive information about how the new change can be used as they normally aren't as familiar with the system as internal parties. The problem statement of communicating changes results in the next use-case.

UC 3: Notify parties of a behavioral change

Event:, A company with different teams, is working on the same product. Teams are divided so they have responsibility for different parts of the system. One team makes a change to their modules and needs to inform involved parties of the changes.

Scope: During development and maintenance there is a need to communicate changes to *all appropriate* parties.

The idea to solve the problems of P1 and P3 requires system level structure of dependencies. With accurate knowledge about the structure of dependencies in a system, it is possible to retrieve every component that depends on and consequently could be affected by a change to a particular module. With this information you can acquire more detailed knowledge about each of these components, including who's responsible for it.

4.2.2 P2: Awareness and P7: Corner case study

Having better awareness and overview of a system will help most problems involving several components and their dependencies. This kind of problems was something that frequently occurred during the interviews. Awareness is a very general problem that can apply to many problems and provide benefit in different ways. Another more specific issue that was encountered is the problem that corner cases involving dependencies that cause problems are difficult to find. Since corner cases are by definition something outside of the normal circumstances, it was difficult to tackle directly. However, better awareness and understanding of how the system works should help in detecting corner cases more easily.

Problem statement 2, Awareness, is about the fact that it is hard to gain insight and overview of a system. This issue is especially true when the system is modular, and the person wanting overview works on a set of modules that are a small part of a larger system. During interviews, it was found that it is not easy to acquire this awareness, especially for new and junior developers as design documents that have been created are rarely up-to-date. Junior developers are often assigned smaller tasks in some specific part of a module. If there were a way to gain a better understanding of the structure of the overall system, problems regarding misunderstandings of functionality would decrease, and it would be easier to start taking on larger assignments faster. Another frequent use-case that was elicited about system awareness was in the case of a major upcoming change. In this event, the problem is that development teams would like a quick way to get up-to-date on the current system's structure to be able to, as smoothly as possible conduct the change. Both these cases were identified as crucial and turned into use-cases:

UC 4: Introduction to a system

Event: A team has just brought on a new junior developer that gets assigned work on a small part of a module. The developer needs to gain insight of the entire system, how his/her module fits into the complete system and the dependencies between its modules.

Scope: During development when a team takes on a new developer or a junior developer is introduced to a system.

UC 5: System overview

Event: A system being prepared for a large change or refactoring process involving one or several teams.

Scope: Lack of awareness may lead to costly problems both during and after a change is made.

Problem statement 7, corner case study, is closely related to overall awareness. However, since it what brought up in several interviews it was deemed eligible for being a separate problem. The difference between corner case study and general awareness is that general awareness should be easier to improve directly and proactively with the help of an updated overview of the system. Corner cases will always be difficult to detect and be aware of before they cause problems. With this in mind, by managing dependencies the goal is to make it easier to detect and be aware of possible corner cases. A use-case of P7 is described below:

UC 6: Corner Case study

Event: A system has the functionality of keeping track and logging data from a sensor. It keeps track of the time and date, when a measurement from the sensor is caught, all of this is logged. The system has been up and running for a long time; it was once discovered that the date was not updated as the time went passed midnight. The system is tested and deployed in a test environment without finding the bug. The bug was, in fact, a fault where if the system needed to achieve data from the sensor in the same moment as the time turned from 11.59p.m. to 00.00a.m. the date was not updated.

Scope: Corner cases can be introduced into a system in any part of development. Corner cases will be dealt with primarily by architects, developers, and testers. This is related to both change-impact analysis and awareness as the corner cases might be easier to imagine with good awareness of how the system works.

The crucial factor to facilitate the problem of general awareness and corner cases is to provide an overview model of the system that is always up-to-date. There should be a focus on providing information on a high level at first, but with the possibility to view details for any parts of the system that is deemed interesting at the time. If the dependencies are continuously updated, it would be possible to use the history of dependencies to show their progress.

4.2.3 P4: Build Requirement

Problem statement 4 is the problem of not knowing what components are required to build a certain module and therefore having to build the entire system every time. UC7, below, was created to further describe the scenario. This use-care were brought up in an interview and described sending over a module to another team for review and the trouble of having to send over the entire system each time.

UC 7: Sending over a module for review

Event: A team is done with the implementation of a module that is one part of a larger system. The module has to be transferred to another team for review. To be sure that the system can run, the entire system is transferred.

Scope: When maintaining a system or during development, developers have to spend much time for transferring software components between teams.

The problem could be reduced as much as possible by knowing exactly which modules are required to try out a certain module. To be able to know what is required to build a certain module based on its dependencies you only need the module in question and every mandatory module that it is dependent on, directly or indirectly. Figure 4.1 shows an example of a dependency model. When B should be built only modules B, C, D, and E are required.

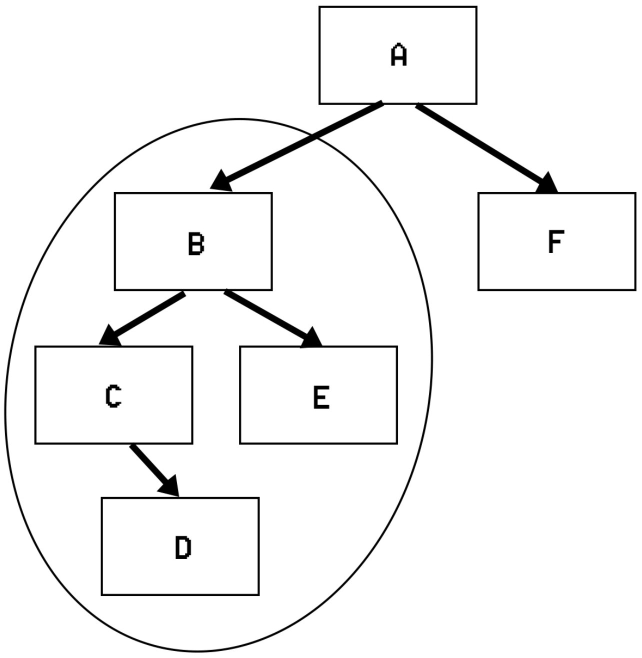

Figure 4.1: Modules required to build module B

4.2.4 P5: Test Optimization

Problem statement 5 involves optimizing testing after a change has been made. From the interviews, it is elicited that there could be improvements in the process of knowing what entities need testing after a component has been made. This information would allow minimizing test times by only testing the necessary parts instead of the entire system. UC8 shows a possible representation of this scenario.

UC 8: Testing after a change

Event: A small change is made in one of the components of a larger system. The developers then need to go through the entire test process, including testing the entire system to make sure that the change works.

Scope: During development, maintenance and testing even a small change has to go through a long process of testing in large systems.

When a module is changed, it needs to be tested with some other parts of the system as well. By using the dependency model, a possible minimal set of components that need to be tested becomes modules that depend on the changed module, modules that are depended by the changed module and the actual dependencies [17]. I can be seen here that not only the components need to be tested, but there is a need to make sure that dependencies work as specified.

4.2.5 P6: Alternative targets

Problem statement 6 about alternative targets were extracted from the interviews in cases where the activity of an alternative of modules is dependent on some other criteria. UC9 shows this in a case where a system has two different modules and which is used is dependent on the screen size. This use-case does not exist in all types or projects as the structure becomes more complex with this type of dependencies.

UC 9: Optimizing performance for different targets

Event: The system is a media player on a smartphone and can be used for both small and large devices. To optimize the quality when rendering an image, two different modules can be used for this purpose, one for large screens and one for small. When the application is installed, only one of these modules can be used.

Scope: There are many contexts where there is an alternative between different modules, e.g. different graphics or settings where the alternative is dependent on hardware or usage of another module. Awareness of this complex structure needs to exist for all concerned.

The dependency model should be able to represent modules that are used depending on some alternative criteria. It should be clear what dependencies are included as alternatives and a prioritization of how they should be used. Otherwise, it is an increased risk of confusion of which one was used during execution.

4.3 Requirements Extraction

As the problem statements and use-cases have been analyzed, ideas that could facilitate the problem statements are acquired in this section by requirement extraction. The analyzing part of the requirement extraction is to analyze the use-cases and extract common and protruding details that characterize the category and can further be used to create definitive requirements. To make sure requirements will exist to cover all of the problems, the problem statements are covered one by one, and the process can be found in the methodology section and figure 3.2. This section will bring forth the high-level requirements that were extracted from the problem. The next chapter, Requirements Design, 5 will show these requirements in more detail as low-level requirements.

It can be seen in all of the use-cases, for every problem statement that they have one major characteristic in common, they refer to a system model of dependencies. Some place where all of the dependencies in the system are gathered. Two items have been identified that are included in the structure, components, and dependencies between them. To represent and visualize this model, components will be visualized as boxes while arrows will represent dependencies, in a dependency graph that resembles the one found in figure 4.1.

One high-level requirement that was extracted was the addition of an "Operational" component to the description of CBS. This is related to PS2: Awareness as it provides a larger understanding and attention to dependencies to another kind of component than software and hardware. This was brought forth as the interviewees wanted the possibility of a component to be dependent on a component that represented an operational part of the development process. One case was a dependency to a legal document e.g. a contract that needed to be signed by a customer or permission access from the user. Another being a module that is dependent on a certain development process aspect, this could be used as a notification saying *"if this component is used there's a need for a different development process"*, could be useful in safety-critical systems where the processes are reviewed and are of large importance. The final reason for an operational component is to connect different customers or external parties that depend on the usage of a specific component. For example, a system can have a different set of components activated for different customers. After a component has been updated, a dependency to an operational component that consists of a list of customers could help in answering the question *who is in need of this update?*. By doing this, a clear dependency between components and customers have been created. To summarize the Operational component, it is meant to be used for operational procedures that could be depended by other components in the system. Above it has been listed three potential uses of this component but the possibilities could reach out from these.

Looking at the analysis in the previous steps it is obvious that at this level there are many different types and traits of dependencies. This is dealt with in different ways in literature which suggest some ideas for categorizing [14] [16] [19]. From the analysis of every problem statement above it is obvious that they would benefit from having continuously updated dependencies. Having a 'version control'-like tool for dependencies requires a defined way of what makes a dependency unique, which means a name and version for each dependency. It is also important to define exactly what a dependency is. Specifically to make a clear distinction between when a dependency should be updated and when a new dependency should be created. From interviews and by analyzing use-cases a suggestion for what should make a dependency uniquely identified is the purpose of the communication between two components. As long as the dependency exists with the same purpose, it should be updated along with the system. If one of two components that have a dependency is changed such that a dependency remains, but exists with a somewhat different purpose, a new dependency should be created and the previous retired. From analysis of the use-cases of awareness and corner cases, there is also a need for tracking progress of a dependency. Each change should have a expressed change-purpose which could work similar to a commit message. It should be noted that a change to a component does not implicate that any of its dependencies must have been changed, an example when the dependencies do not change is when refactoring a code fragment to optimize the lines of code but keeping its original functionality.

Since every dependency is represented in the structure mentioned above, it is possible to give each dependency attributes to represent its traits. The P4: Build requirements and P2: Awareness suggests there is a need to know if a dependency is mandatory for building or not. To respond to the use-case of P3: Communicating Changes to either internal or external parties, this could be a suitable attribute as well.

A special attribute that would greatly improve the categorization and the ability to involve different dependency is to give each attribute a type. This was considered early on in this thesis work as well because one goal was to be able to generalize the handling of different kinds of dependencies. Categorizing dependencies in types have also been done in previous literature [19]. The types that will be suggested in the requirements should represent an appropriate set of types to facilitate with the problem statements and use-cases. To facilitate with P1-3 and 7, a better understanding of the system is required. To acquire this understanding dependency types will be used in this thesis as well. Common dependencies of data and control should be included [5] and to accommodate P6: Alternative Targets an alternative type of dependency could be included.

The following is a summary of above-mentioned requirement extraction and reasoning behind them for managing dependencies to attempt to solve the acquired problem statements.

- A dependency model is required to represent the components and dependencies that make a system. The model should be able to represent both components and dependencies.

- A component called "Operational" should be added to the model in addition to regular software and hardware components. The purpose of such a components is to represent an operational part of the development process. Examples of what could constitute an operational component is a legal document, a certain development process or a subset of the customer base.

- The dependency model should be continuously updated. It should also be possible to track and view the historical changes of dependencies in a simple way. The trade-off between effort and value for a developer using dependency management tool suggested should be highly prioritized, i.e., many interviewees do not mind the tool being basic if it means it is easier incorporated in their daily work. Many tools are being used as it is already.

- The model needs a precise definition of what constitutes a dependency helping to acknowledge when a dependency should be either updated or replaced.

- It should be possible to categorize dependencies based on its type and an appropriate number of types should be used considering complexity vs. value towards solving problem statements. It should also be possible to easily ascertain whether a dependency is mandatory or not, as well as connected to an internal or external component.

5 Requirements Design

Based on the analysis of chapter 4 this chapter will present the detailed requirements ideas that are the result of the work for this thesis. The chapter consists of three sections that together describe how our research suggests dependency management will be done to solve the stated problems. The first section will contain requirements of documentation and processes required to conduct the intended dependency management. How the information about dependencies and the related components should be stored, updated and their life cycle. The second section will present the dependency model and dependency structure. What is considered in the scope of dependency management and the attributes required for a dependency. The third and final section will present requirements for functionality for a tool that would be based on the dependency management ideas suggested in this thesis. Presenting the functionality as requirements will provide a clear and concise presentation of the results derived from interviews and previous research and analysis.

5.1 Documentation & Process Requirements

As the definition of a dependency that is found in section 2.1 is quite wide and imprecise a dependency can exist in many different variations, some of which have been described throughout the report. One of the things that were extracted in section 4.3 is that every dependency should have a purpose of why it exists, because if no one knows the purpose of why a dependency exists it probably isn't necessary or there's a need for better understanding of the system. If two components depend on each other with multiple purposes, these shall be divided into different dependencies. This brings out the first requirement.

REQ1: Every dependency shall have a purpose.

Research Question 2 *(How can these dependencies be documented and handled from a configuration management perspective?)* Together with the possibility of following the evolution of dependencies within a system that arises in UC 4-6 and the problem of awareness creates a motivation to keep track of dependencies and being able to trace dependencies to a specific version of a system. An already existing practice at the companies that were interviewed was version control and by including dependencies as configuration items (CI)(see [3] for definition) it provides the possibility of storing and keeping track of dependencies. By including these into version control, they are given a version which is increased after every update that makes the evolution easy to track. There are two different aspects that is interesting of following the evolution of, each individual dependency and which of the dependencies that are included for a specific version of a system (a system's *dependency model).*

REQ2: Each dependency shall be included as a CI of the project.

By including dependencies as CI's one of the aspects are taken care of. To be able to connect a version of the system to a specific set of dependencies a dependency model is created. The alternative is to connect a dependency to a specific version of the system when committing a change to a system a majority of the dependencies often keeps unchanged (unless some major refactoring has been made). This is the reason why inspiration taken from the CM-model, composition model [11] was used for creating the dependency model. This model contains a connection to what version of the system it belongs to and a list (with name and version) of all of the dependencies that are a part of this system. This is to make it possible to find the set of dependencies that belong to a specific version of a system.

REQ3: A system's *dependency model* shall be included as a CI of the project.

The system dependency model can be used differently for different projects, but its purpose is always to describe what dependencies that are included in a specific version of a system. Because of the difference in how different projects use version control to perform configuration identification (shortly described in section 2.3), it is decided to leave naming and how versions shall be defined to the project owners and left out of scope to this thesis. In some large projects might not be feasible to create one dependency model for each new version of the system as this amount would become too many to comprehend. To create a reasonable amount of different versions of the dependency model a new version is only created when the set of dependencies change. If there is no dependency model to a specific version of a system, the dependency model to the most reason version is used. An example of how the dependency model can be linked to different system versions can be found in figure 5.1. In step 1 in the illustration, a dependency model (version 1.0) is created that includes all dependencies that exist in the system (version 2.0.1). In step 2 the system is updated to version 2.0.2, but there is no change in dependencies, therefore the same dependency model (version 1.0) is used. After updating the system to version 2.0.3 there is a change to a dependency and the dependency model is therefore updated to version 1.1 and now includes all dependencies that are included in the system (version 2.0.3).

Because dependency identification is left out of the scope of this thesis, there is still no automatic way of finding a complete set of dependencies within a system. To still keep an updated collection of dependencies requirement four is created that is related to handling dependencies directly

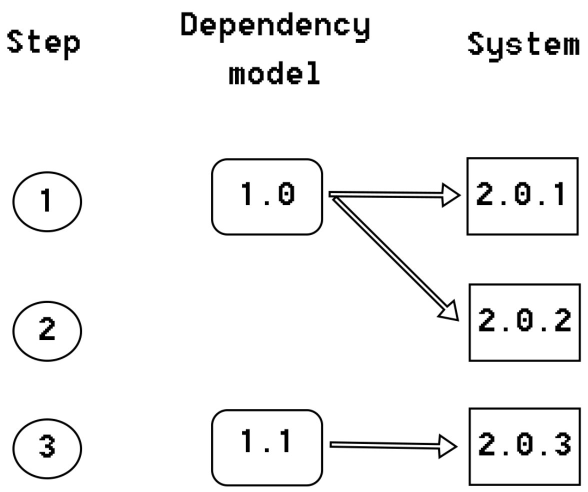

Figure 5.1: How a dependency model could be linked to system versions.

as they are identified.

REQ4: If a dependency is identified in the system that does not correspond to the documentation, the documentation shall be updated, so the dependencies resembles how the system works.

Requirement 4 works as a notification to start the process of creating or updating a dependency. To create a standardized and controlled process around dependency management, a lifecycle of a dependency is created that can be seen in figure 5.2. When creating a dependency the identification in figure 5.2 is realizing the need for a new dependency (REQ4), and the next step is to define the purpose of *why* the dependency is needed (REQ1). The purpose will be one of the attributes of a dependency, which are defined as requirements in more detail in the next section, 5.2. After documenting these attributes, it's run through a quality assurance (QA) process including *all related parties* to make sure that everyone that has a connection to this dependency agrees of the purpose and the attributes that have been set. Who's included in all related parties' is decided by the administration of the project but it could preferably include e.g. project manager, the responsible for the related components and experts from the different teams to make sure that the dependency is correctly documented. If this is not the case, it will go through the process again. Otherwise, it will notify every party that is in need of the notification and the dependency is ready for utilization.

REQ5: The lifecycle of the dependency CI shall follow figure 5.2.

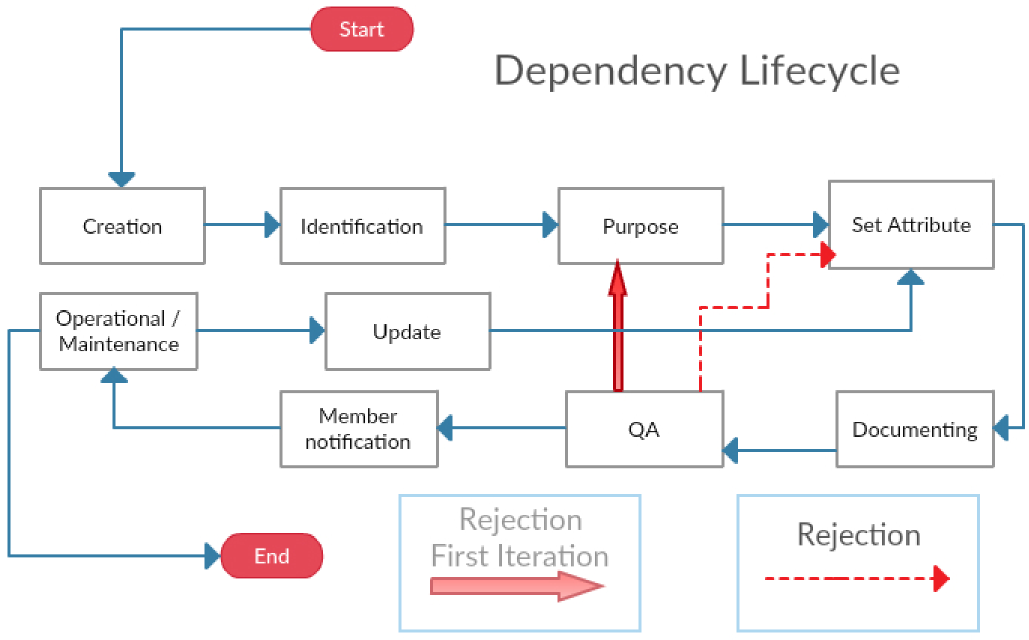

Figure 5.2: The lifecycle of a dependency

5.2 Dependency Structure

From the interviews, it is clear that there are many different kinds of dependencies to be managed. A dependency between two components does not have to exist only for one specific reason, but they can depend on each other in multiple ways. Instead of creating many different kinds of dependencies between two components in a dependency model where each one provide a specific quality of the relationship it is more convenient to make a dependency consist of several attributes that define the qualities of all dependencies between the two components in question. This also makes it easier to oversee as there is only one dependency that describes the relationship between two components instead of several independent ones. These qualities are hereby defined as attributes and are essential to utilize dependencies to deal with the stated problems for this thesis 4.1. Table 5.1 gives an overview of the attributes of a dependency. Attributes marked with * are the ones that should be considered for version control, further explained in subsection 5.2.1. Further below each attribute is described more in depth to form a dependency structure.

Dependency Name	Connected Components*
Responsible	Purpose
Utilization*	Reason of Change
Version	Mandatory
External	Type

Table 5.1: Attributes of a dependency

At the beginning of the analysis shows that it shall be possible to identify a dependency. Like any other file or item involved in software development, a dependency needs a name for identification. There are several different naming-conventions in regards to software development, but the important things is that a dependency is easily identified, both by a tool but also by developers. The latter to contribute to better communication around dependencies as it becomes easier to communicate around dependencies. The specific process revolving configuration identification, including naming-conventions, are out of scope for this thesis as motivated in the previous section (5.1).

REQ6: *Dependency name.* A dependency shall have a uniquely identifiable name.

Since a dependency is a connection between two components, a start node, and an end node, where the start node depends on the end node, e.g. if a module, Component A, depends on functionality in a library, Component B, Component A will be referred to as start node. An attribute stating both these components must be available in a dependency.

REQ7: *Connecting components.* A dependency shall have information about the two components it is connected to. Referred to as start and end node.

Requirements extraction 3.3.3 shows that there is a need for knowing who should be notified when a component is changed. Meaning that if Component A is dependent on by Components B, C and D, it should be possible to easily notify the appropriate party for each of those components that a change will be happening. The appropriate party could for example be the team leader, architect or the responsible for each component. This was extracted primarily to assist UC3: Notify parties of a behavioural change but with the knowledge about whom is responsible for a dependency may aid in more of the communication related problems e.g. knowing whom to go to if a question about a dependency arises.

REQ8: *Responsible.* A dependency shall state a responsible. The responsible shall preferably be someone who is responsible for the start node component and can therefore be notified if changes are made to the end node, which by definition will effect the start node.

As described in the previous section and REQ1. Every dependency needs a purpose, and it is appropriate to store as one of the attributes. This is because the purpose is always connected to one dependency.

REQ9: *Purpose.* A dependency shall have an attribute describing the purpose of the dependency. It should describe at a high level why the dependency exists.

A major benefit with dependency management for all use-cases is the fact that dependencies are continuously updated. The following three requirements describes attributes that will be continuously updated, helping to create awareness and assisting with all use-cases but primarily use-case 4: Introduction to a system and 5: System Overview.

REQ10: *Utilization.* A dependency shall an attribute describing the current functionality of the dependency. How it works at a more detailed level than REQ9: *Purpose*, for example, API-calls. The utilization is updated along with the dependency.

REQ11: *Reason for change.* For each update of the dependency, a reason of change shall be entered along with the new dependency.

REQ12: *Version.* A dependency shall have a version number which is increased for each update

Analysis and requirements extraction describes that there is a need for knowing whether a dependency describes communication between components that is mandatory for building the system or not. This was elicited from interviews that described a need for trying out a module in a system; this module might depend on 5 or 6 other components to build and run. But without a dependency model, it was hard to keep track of which components and therefore the entire system with over 100 components had to be fetched, built and run. This is further explained in Problem Statement 4: Build requirements and UC 7: Sending over a module for review.

REQ13: *Mandatory.* A dependency shall have yes/no attribute whether the functionality of the dependent module from this dependency is mandatory for the system to build.

During analysis it also became clear that it would be beneficial to include not only dependencies to components developed internally but also externally. Knowing that a dependency is connected to an external component will especially benefit UC3: Notify parties of and update and UC 4 and 5 involving awareness. The reason for separating external and internal came from interviews and the desire to be able to adapt the work process depending on the situation. For example, if a dependency is to an internal component there might be an informal and quick communication about changes but if it is external a more formal process might be favorable.

REQ14: *External.* A dependency shall have yes/no attribute that describe whether the dependency is connected a component where the responsible is within the company or external.

Analysis of both interview material and previous research show that to provide a flexible dependency structure that can deal with the multiple problem statements in section 4.1 there is a great benefit being able to categorize dependencies. Therefore a dependency shall have an attribute called *type* that specifies what types that a dependency belongs to (this can in some cases be more than one). To clarify, a component can be of for example both data and control dependency, specifying the relationship of how one component depends on another. Types are described in table 5.2. *Data* and *Control* type are the classical software dependencies and are therefore obvious choices to include [19]. *Sequence of flow* was inspired by R.L. Nord et al. and its purpose verified in the interviews for this thesis. The *Alternative*-type is introduced for this thesis to solve PS6 and UC4 where a component is dependent on one out of a group of components dependent on some criteria. Together with the type alternative information should be stored in the dependency what other dependencies are in the same group. As well as what prioritization this dependency has in the group. This will give a tool the ability to visualize and in other ways use information about the alternative. *Hardware* is included to be able to include some crucial hardware components into managing dependencies. Finally, the *Configuration* type is introduced to give some flexibility to the dependency model and categorize dependencies between components that depend on each other for various reasons (Further described in 5.2. Common for these various reasons is that they are all things where the dependency describes something that must be in a certain state or have a certain status or value for the system to work.

Type	Description:
Data[19]	A data dependency describes one module being dependent on the data produced by another module.
Control [19]	A control dependency describes one module being dependent on the existence of another module. To for example make calls to its API.
Sequence of flow [19]	A module is dependent on another module having executed something before it can execute itself.
Alternative	A module is dependent on one out of a group of several other modules. Information about the prioritization of this dependency in the group as well as what other dependencies are in the group shall be listed as well. A dependency of type alternative will also have at least one other type to specify what kind of dependency it is.
Hardware	A hardware dependency describes a module being dependent on a hardware component
Configuration	A configuration dependency describes a module being dependent on the setup configuration of another component. This could be a module being dependent on that another cannot execute in the same memory or processing core and therefore depends on how the other module is configured. Another example of a configuration dependency is a module being dependent on an operational component representing a development process or a legal document.

Table 5.2: Dependency types with description (REQ15)

REQ15: *Types.* A dependency shall have a type attribute that describes its type which can be one or several out of the types described in table 5.2.

5.2.1 Version Control consideration

Out of the ten attributes found appropriate for a dependency, two of them should be considered to be under version control. Meaning when these two attributes are changed, a new version of the dependency should be created. The first one being *Utilization* (the more detailed description of how a dependency is utilized). Every time the utilization of a dependency is changed, but the overall purpose remains the same, a new version of the dependency should be created. The second attribute considered for version control is *Connected Components*. If for example, Module A has a dependency to a library with a certain purpose. Developers can choose to change the library but Module A communicates with the new library with the same purpose as before. Then a new version of the dependency should be created but with the end component set to the new library.

Dependency Name and *Purpose* should not be changed and are therefore not considered for version control. The *Responsible* attribute is more like metadata to help with communication and should not be considered either. The attributes *Mandatory*, *External* and *Type* all describe some aspects of the dependency but from the interviews has risen that these should not change without changing *Utilization* or *Connected Components*. Therefore *Dependency name, purpose, responsible, mandatory, external and type* will not trigger a new version of a dependency.

Finally, attributes *Reason of Change* and *Version* can be considered as effects of version control, *Version* being updated for each new version and *Reason of Change* similar to a commit message explaining the reason for an update.

This setup of version control and what attributes shall trigger a new version of a dependency is a model set by the authors of this thesis without larger research nor validation. This is a reason why this subject needs continued analysis and work. This will be further discussed in chapter 6.

5.3 Functionality Requirement

Some functionality requirements have been extracted that can work as a basis for a possible tool implementation but because of the scope prioritization of solving the problem statements in section 4.1 above creating a requirement specification for a tool, only functionality requirements that are connected to solving the problems will be presented.

The visualization will be a big part of especially PS2: Awareness and to gain an understanding of the system. There are many visualization techniques for dependencies, matrices [17], tables [19] and graphs [14][22] [4]. Graphs were chosen because of providing understanding of overall relationships between components and it's a good way of finding root-causes to a problem [4]. It's hard to get an overview of the relations in a table while it's hard to store and visualize attributes in a dependency matrix.

REQ16: Components and dependencies shall be able to be visualized with a dependency graph. Similar structure to the one shown in figure 5.3(a).

To be able to get an overview of what components and dependencies that are included in the system, the graph will present the name and versions. This basic overview that could help in UC5: System overview and UC6: Corner case study due to the lack of system understanding mentioned in the analysis.

REQ16.1: The dependency graph shall show the name and version of components and dependencies.

In UC4: Introduction to a system there the goal of not only getting the overview of a system but also creating a good understanding of the relationship between components. By visualizing the components and the purpose of why each dependency exist it will provide a better understanding of the

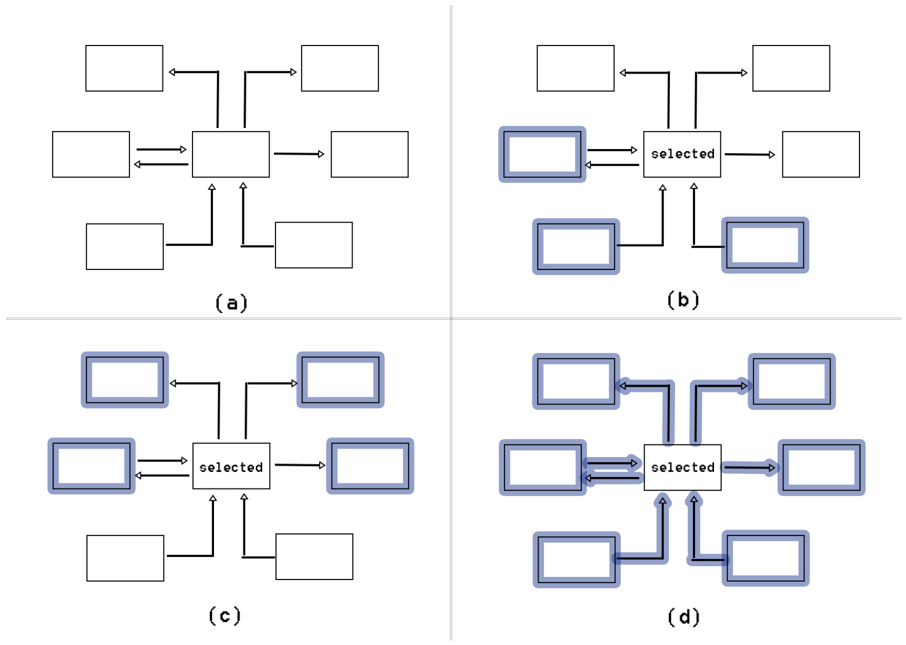

Figure 5.3: Presented examples as dependency graphs.

dependencies within a system. The purpose and type of a dependency can aid in UC 2: Analyzing the impact of a change by creating a context around why they are related and what kind of problem it might cause in affected components. These attributes are not a part of the basic view of the graph as they are not always needed to the same extent as name and version.

REQ16.2: The dependency graph shall be able to show name and version of components and the purpose and type of the dependencies.

If more information about one specific entity (component or dependency) of the graph is needed, it shall be possible to obtain the related data. E.g. when looking into the use-cases of PS2: Awareness there are situations where it's needed a complete set of attributes to understand why an error has occurred, or calculating the change-impact of an upcoming change.

REQ16.3: When a component or dependency is selected in the graph it shall present detailed information of that entity in form of its attributes.

Some problems, still regarding PS2:Awareness, that arose during the interviews is the evolution of a dependency which created REQ17. Because of putting dependencies as CI's it's possible to find a history of how that dependency has evolved. With traceability of the evolution of a dependency and the dependency models, it's also possible to see what dependency related changes that have been made between two versions of a system. The authors believe based on the interviews and ideas taken from some dependency application sources ([6] [17] [22]) that to be able to find what dependencies between two versions that have been changed and history of the dependencies could be useful if an unknown error has occurred.

REQ17: It shall be possible to present a list of dependencies that was changed from the previous version of the dependency model and in detailed view of a dependency get the history of older versions.

UC1: Pre-change analysis of a system, UC2: Analyzing the impact of a change and UC3: Notify parties of a behavioral change are all in need of functionality to look up which components are dependent on a selected component. For UC1 it is important to know what is dependent on a component and with what types of dependencies before a change. For U2 a list will help track the potential error. For U3 knowing what is dependent on a component after a change has been made will help communicate those changes. The components that depend on a selected component have an arrow to it, an example of the set of components are highlighted in figure 5.3(b).

REQ18: It shall be possible to acquire a list (with name and version) of all components that depends on a selected components and what type of dependency that is used.

Since one of the types are Alternative, describe in table 5.2, if there are alternative type dependencies to components in the list it should be presented which alternatives belong to the same group.

REQ18.1: If components in the list provided by REQ18 are of the type Alternative, it shall also show components that belongs in the same group.

For a tool to assist with the problem about build requirement and UC7: Sending over a module for review there is a need to easily acquire information about what components a selected component depends on (an example is visualized and highlighted in figure 5.3(c)). This to easily know what modules are required to build and try the selected component.

REQ19: It shall be possible to acquire a list (with name and version) of components that a selected component depends on.

There shall be functionality to acquire the same list as in REQ18 but only showing the components that are connected to the selected component, directly and indirectly, through mandatory dependencies. This is needed to easily identify the minimum set of components required to try out a selected module, again related to UC7.

REQ19.1: It shall be possible to acquire a list (with name and version) of mandatory (REQ13 with status *'Yes'*) components that a selected component depends on.

To deal with problems about testing like UC8: Testing after a substantial change the tool shall be able to return a list with all components connected to the selected component. As well as the dependencies that connect to them. Example highlighted in figure 5.3(d). This because it is not enough to test each component that is connected to the selected component but also make sure tests are done to test the actual dependencies/communication between components/integration tests as well.

REQ20: It shall be possible to acquire a list (with name and version) of all component that depends on a selected component, components that the selected component depends on and all dependencies linked to the selected component.

6 Discussion and Related work

This chapter will discuss some of the phases of methodology for this thesis as well as its results. It will start off by bringing forth some of the thoughts about the strengths, weaknesses, and lessons learned of the resulted methodology and how this derived from the one that was planned when beginning this process. After that, some of the validated aspects of the result will be presented. This section will also be answering questions of *what was acknowledged during the validation?* and *what is missing with the results and are there any possible new results that could be complements to the requirements?*. This discussion will revolve mainly around ideas that have emerged during or after the requirements were set and validated. After discussing the results, some aspects that can be seen as threats to validity will be presented.

To present how this research relates to others regarding scope and findings there will be a section that presents some related work. This section will discuss some of the aspects were the related research and similar findings and where they differ. At the end of this chapter, some ideas that can be used for possible future work in the area of dependency management will be presented.

6.1 Evaluating methodology

One major difficulty with managing dependencies is how dependencies are identified and to recognize changes in them. This was thought of early on in this thesis and strengthen by pre-analysis of previous research [19] [10]. Today there are several ways of identifying dependencies, but they are often language specific and works on a detailed level by code analysis. Since research question 1 was to elicit the core problems around dependencies between components, there was an incentive to gather information from several different companies and projects. This meant not having a specific project in mind when managing dependencies and less value in finding suitable identification tools for each project, which would have taken a considerable amount of time. This lead to identification being left out of the primary scope of this thesis and the goal set to exploring dependency management in a generalized context, leaving it up to each project to realize their type of dependency identification. Furthermore, since this thesis set out to manage dependencies in a generalized way some presented types of dependencies can not easily be identified automatically. For example Sequence of flow and Configuration dependencies. Validation interviews showed that a manual identification of dependencies can still be a viable solution if the dependencies that are managed are high-level and thereby within a reasonable amount.

An important part of this thesis was the interview process used to elicit information about the problems around dependencies. The methodology of this process, described in section 3.2.2, was set as a range of three different types of companies and project managers, configuration managers, architects and developers to get different viewpoints of the problem. This methodology was realized as planned except for not talking to an as wide range of companies as intended. There were nine interviews in total at three different companies and some independent people. The number of interviews was appropriate although more is always better. It would be better if these interviews were spread out over more than three companies to get a better to get a wider spread of input. That would be better to assert that the found problems exist at more places and to assert that as many problems as possible are considered.

To explore dependency management in a generalized way a lot of time was spent during this thesis to gather and explore a lot of previous research and different company contexts. In hindsight, this methodology gave a solid foundation and understanding of the subject and its applications, but it did leave less to get an in-depth solution, for example, a proof-of-concept. If instead the thesis only had one software project in mind it would be possible to show dependency management in practice. The more generalized route was chosen for two reasons. It made more sense to look at the problem generally before trying out a practical solution. It also seemed like a hard sell to a real software team to spend time trying out something rather unexplored.

This choice to explore rather than go in-depth in a practical setting also affected the collaboration with Softhouse Consulting and the mentors for this thesis. During the first two months, there were continuous solid contact and consultation with Softhouse. The problem of dependency management is of course not a problem exclusive for Softhouse, but their consultants had detected the problem at many different clients project and therefore worked as a stepping stone for this thesis. The consultation with the mentors at Softhouse helped clarify the subject of dependencies and guide this thesis to a path that had a possibility of leading to results. After the initial two to three month when the interview period was over, and analysis started, the communication with Softhouse declined. If this thesis was to be done again, it might be preferable to keep a more steady communication to exchange ideas not only in the first half but also during the analysis and validation phases. The reason communication declined was because analysis took a lot of effort as well as preparing and partake in the SNESCM-conference, conducting validation and not to mention writing the report.

There were a lot of different approaches to how one best would present the results in this thesis where the decision finally landed on presenting problems, use-cases, and requirements. The problems were presented because research question 1 was to elicit the problems. They did end up being very general and not so self-explanatory therefore they were complemented with use-cases to give further description of them and their context. The choice to present the solutions to the problems through requirements was made mostly because the initial idea was to show the result with a simple tool as a proof-of-concept. The motivation for wanting to create a tool was to show the results of a general and abstract subject in a practical and concrete way. But the deviation from making a tool was necessary because it became clear that it was an unrealistic time cost to build a decent enough tool and also such a result would be very context dependent. Instead, presenting requirements for a tool was a way to show the results in a similar way as a tool would have.

As mentioned in section 3.4 validation was primarily done in two different ways. Qualitative interview and a workshop during the SNESCM-conference. The workshop wasn't in the original methodology since the opportunity to partake in the conference appeared after the planning for this thesis was done. The workshop took place when initial findings and some results were set. Meaning there was an opportunity to get initial validation on these. As far as the setup of the workshop there was little room to changes from, a small introductory presentation was done to advertise the thesis and then the one hour workshop took place. A lesson learned from the workshop is that one hour goes by quickly when ten interested people are to discuss a wide subject. In hindsight, it would have been even better to narrow down the discussion to two or three key points.
In regards to the validation interviews, they were intended as qualitative interviews with discussion as mentioned and motivated in 3.4. The goal was to perform 3 of these interviews, but as they had to be held in July, there were scheduling difficulties. It ended up being one longer validation interview which went through most results and ideas presented in this thesis. It resulted in good feedback and especially brought up valid concerns about some results that would not be viable in a large company setting.

A larger scale validation would bring more validity to the results but the choice to not make a questionnaire type validation still seems reasonable. A questionnaire around this type of theoretical results would require 20 pages of explanation before asking questions, and that would be an unfeasible cost of time for limited extra value.

6.2 Validation of results

After analysis and requirements extraction the results were presented and validated through another interview and the SNESCM-conference with people from both industry and academia, further described in section 3.4. Overall during the validation, it's been elicited that dependencies have been noticed by most parties, but the idea of generalizing dependencies, managing and utilizing them to this extent is a new angle to solve the found problems. The dependency related problems that came from the interviews were highly correlated to problems that were found during the literature search. Every problem was also something that validation acknowledged and could relate to. In fact, there was only one problem that wasn't found in any literature but existed in two of the three interviewed companies, Problem 6: Alternative targets. These following paragraphs discuss around information gathered from the validation (interview or SNESCM), and the last one will bring forth some aspects of threats to validity.

One of the most interesting areas of the validation interview is if our way of dependency management would be possible to implement in the context of a company. Even if the main principles and utilization areas would have been possible to implement there is one problem that has risen. To be able to rely completely on the different utilization areas there is a need for a large percentage of the existing dependencies to be identified and put in the dependency model. Otherwise, if the dependency model isn't complete some of the functionality requirements (presented in section 5.3) isn't complete this might become more of a burden than an aid. This brings out a need for a process around how to introduce dependency management into a project. This process is something that needs to be researched, tested and evaluated but a proposed process could be the following. When introducing dependency management into a new project, this problem does not occur as long as dependencies are documented continuously as they're created or changed. If dependency management is introduced to an existing project with a lot of undocumented dependencies a suggestion is to have an introduction phase where people responsible for a component has responsibility for creating the dependencies their component has. Even after this phase, it will probably exist some dependencies that will be added as they are identified, but it will provide a set of dependencies to start utilizing. Note, however, that if in the future identification of these kinds of dependencies is automated, an introduction phase won't be necessary.

The most essential problem was the problem of creating a correct change impact (P1). By utilizing dependencies, it's possible to find what components that a change may affect [6][21]. With the information in the attributes about the *purpose*, *type* and *utilization* of the dependency it was suggested in the analysis and acknowledged during validation to provide a deeper understanding of how a change might affect other components in a system. As this change-impact analysis can be utilized both before and after a change, its purpose can be used in most projects, if not as a part of pre-change control as a part of finding the set of components where a change might affect.

Everyone that was contacted during this thesis was aware of the existence of dependencies between components in their systems but not to a large detail. One of the main advantages with dependency management is the increasing awareness in not only that dependencies exist but *what* dependencies exist. During the SNESCM validation it was mentioned that knowledge of dependencies within a system is known but often by a limited amount of people, and due to not documenting dependencies it's hard to spread this knowledge. That the knowledge of dependencies exists only strengthen our findings, with the possibility of documenting and visualizing dependencies awareness of both dependencies and *how* a system works can be accomplished and spread. A possible new result was also mentioned during the validation of one aspect that could aid in the problems revolving awareness, how components are connected to functionality (or specific requirements in a requirement specification). This was left out of scope earlier in the thesis but will be seen as future work (motivation can be found at the end of section 4.1). But it is believed to open up many possibilities to aid in awareness related problems, including coordination between teams and give more insight to management fields (like requirement handling). During validation, it was extracted a new requirement that would aid in the problem of awareness, the attribute of "linked dependency". This would give the related dependencies with a definition of *if one changes, the linked dependency changes as well.* These dependency 'pairs' exists in software and being aware of them in a dependency management tool would be a great addition to the existing requirements.

In some aspects awareness and communication are highly connected. It was validated that by creating a generalized way of documenting dependencies it enhances awareness but also creates the possibility of communicating around dependencies with names and attributes. Almost every developer has encountered communication problems within a software developing team and managing dependencies might be a step in the right direction by adding more possible ways of directly communicating around the behavior of a system. If components are developed by different teams, it also gives a quick way of finding whom to contact when a change has been made. Though, in a large system, it might be many people that want to be notified of a change that isn't responsible for a dependency. This creates a complementary result of a functionality of adding additional notification participants to a dependency.

From the validation interview it's elicited that to use the test optimization functionality there is a need to automate it. This creates the need for linking each test-case to either a component or dependency and integrating testing into a tool that only runs the necessary tests. During validation, it was agreed on the entities that needed testing after a change. This knowledge is based on a paper which future work also includes creating a tool and test in a cooperative environment, so there is a need for further verification and testing of this knowledge before seen as complete.

An aspect that the authors did not find a similar solution too is the awareness and management of alternative targets. The functionality of how the system works needs to be structured and handled in the code. Validation strengthened that by adding a type, *Alternative*, creates a more complete and accurate view of the system and helps in some aspects of awareness and change-impact as it becomes possible to see when and why an alternative component is used. So as a clarification, the requirement of an alternative type does not facilitate the implementation of alternative components, only makes it possible to visualize and increase awareness of its existence.

The problem of finding out corner cases does not have any set solutions to it. During the first iteration of interviews, it was discussed that with a complete and correct set of dependencies it would be possible to predetermine all possible corner cases as every relationship between components are documented. During validation, it was elicited a complementary result of integrating a new attribute of limitation. This attribute would keep track of the limitation of a dependency has to know *When does this dependency fail?*. This information together with purpose and utilization of surrounding dependencies would create one opportunity of investigating how one component might be used to limit another.

One problem that was given to the authors before the interviews to start with was the possibility of checking consistency within a composition of components. Functionality was thought out based on a tree structure of components and logical values of *AND, OR* and *XOR* dependent on the dependency between components were a primary idea of solving this problem. During the first iteration of interviews, it was never mentioned that a similar problem had risen and due to the methodology of extracting problems that were encountered at the contacted companies this was not a problem in need of solving. Therefore not mentioned as a problem in this thesis.

Some of the requirements stated in the last chapter have some aspects to it that are quite abstract e.g. procedures correlated with CI's, QA and Operational/Maintenance in the dependency lifecycle. Dependency management has the possibility to benefit many different kinds of companies that use different company structures and processes. The idea was that the flexibility of customizing how these aspects will be performed to fit the contextual company benefits in how easy to adapt dependency management will become. This flexibility can also create uncertainties in the aspects that can hinder the utilization of dependency management

As mentioned throughout this thesis it is difficult (but not impossible) to detect all mentioned dependencies automatically. This leads to a disconnection between the system that is being worked on and the dependency model that describes that system. If future work can figure out how to identify new and updated dependencies 100% of the time this disconnection won't have to exist. Apart from just being less accurate, having a disconnected dependency model leads to difficulties if changes are made to an older version of the system and updating the dependency model along with those changes. There will be no suggested solution to this problem in this thesis just a recognition that it exists and is a downside of managing dependencies along with a system.

The results of this thesis have been validated through two phases with different people to a different extent. An aspects of threats of validity is that it was hard to schedule validation interviews. This lead to only one interview being held with only with one person (from one company), this only gives new insights from one part, leaving out many possible aspects and comments on this work. Related to this is also that the interviewed person is someone that was used in the first iteration of interviews, therefore the result is partly based on the answers that they gave during the first interview. It was intended to carry out 2-3 interviews where preferably 1-2 would come from companies, new to this thesis.

6.3 Related work

Trosky B. Callo Arias and Pieter van der Spek looked into this when they did a systematic review of dependency analysis [7] with the conclusion that awareness and utilization of dependencies shows a lot of promise. Which was encouraging when starting on this thesis. In their review, it was believed dependencies could be categorized as either, structural, behavioral or traceability dependencies. In this thesis, there are more types of dependencies presented, but there are several similarities between the types in this thesis and the structural and behavioral of Callo Arias and van der Spek. On their level of dependencies, they found that impact analysis, system understanding, traceability and fault localization are some of the areas that can benefit from dependency analysis. Which further strengthen this thesis claim that impact analysis and system understanding are some of the core problems. The review points out that no existing technique is capable of identifying all kinds of dependencies, but they present the kind of dependencies that each technique identifies.

Robert L. Nord et al. did an in-depth study on a safety-critical system that had to undergo extensive refactoring due to the increased complexity in finding root-causes of a problem [19]. A dependency model of the system was created by examining how system components interacted. This led to alternative ways of presenting information about dependencies built upon what it should be used for which lead to the need to categorize dependencies into types. Categorizing dependencies was something thought of early on in this thesis as well. Although Nord et al. categories are many, they don't use all of them. Their categories are also at a more detailed level because their primary need is to use the dependency model for root-cause analysis. This thesis has categories that are on a higher level to satisfy the issues of for example system overview and communication. One of the motivations for their study is that *"Research continues to focus on more tooling and automation to assist with dependency analysis rather than interim, easier-to-adopt solutions"* [19]. This claim is something that this thesis acknowledges and adds substance to managing dependencies.

In a doctoral dissertation by Cleidson R. B. de Souza from 2005, an empirical research of software dependencies resulted in the design of a tool called Ariadne [10]. De Souza claims that the problem of dependencies between components is more of a problem about coordination than it is about software architecture. Which is something this thesis have embraced as well. De Souza also claims that there are two prominent limitations to managing dependencies. The first being identification of dependencies which is something this thesis acknowledges as well. The second being that research often look at dependencies as a framework and not from the point of view of a developer. This is something this thesis has tried to deal when suggesting functionality by taking into consideration developer needs and limitations. The tool, Ariadne, aims to facilitate managing dependencies, for software developers. This tool is used specifically for the Java language and is accessed as a plug-in for the Eclipse IDE [13]. The limitations of existing tools are brought up as lack of facilitating cooperative efforts, e.g., answering the question: *what changes affect me?*.

Besides language specific tools as Ariadne there are som conceptual dependency analysis techniques. Judith Stafford and Alexander Wolf developed one of these called *chaining* [23]. This paper lifts the difficulties with a general, automated dependency analysis on a system model made from an architecture description language. The technique is built on applying some algorithms to an architectural description that is inserted by the user. This is one similarity that has been found in most of the conceptual techniques including the work made in this thesis, the need of manual input and knowledge from a user. The paper takes use of three types of chains that describe different relationships between elements: affected-by, affects and related.

These are the same groups of relationships that are utilized to solve some of the problems of this thesis e.g. change impact and communicate changes. The paper and this thesis both have the idea of creating a concept of managing dependencies in a way that is not programming language specific but as the analysis in the paper creates a system model from another type of language it creates another language dependency.

When visualizing dependencies there are many alternative techniques to choose from. Bixin Li et al. has written a paper on a categorization of the existing dependencies and the qualities and applications of representing them in a matrix [17]. The categorization (or types as it's called in this thesis) is similar in some aspects, but the paper creates a set of types that represent relations that can exist through an open interface. Findings from the interviews motivated the wider approach in this thesis because of the many problems that arise in relations other media than open interfaces e.g. protocols or software-to-hardware communication. There are many advantages of being able to quickly determine the relation-classes of relates-to, depends-on and depended by. One of the difficulties of using matrices is representing a large amount of metadata that is connected to the dependency. The paper only gives a dependency one attribute (type), and the large amount that was added to the dependency structure of this thesis motivated choosing graphs for representation instead of matrices. It could be motivated to combine the two techniques for systems with plenty of dependencies, visualizing dependencies with the help of a graph and doing calculations of relation-classes from a corresponding matrix.

6.4 Future work

This thesis has presented an overview-oriented way of performing dependency management. The area of dependency management is relatively unexplored and has a lot of potential in development. Most of the future work has been mentioned throughout the report as a part of out-of-scope problems due to the time limit of the master's thesis; others are ideas that were thought of too late in the process without time to take it into consideration. Some of the future work include:

- Integrate this process into the context of a company for further testing and validation of the results. Since this thesis is general and theoretical a good way to further validate the findings and analyze the results would be for a future thesis or other interested to try out this type of dependency management in a real project.
- Implement a tool that handles storing, documenting and utilization of dependency management. The theoretical nature of this thesis will also work well as background material and a stepping stone for building a tool with the functionality described in this thesis.
- Investigate the best way to integrate dependency management into an existing development process.
- Find the connection and connect dependencies to functionality and requirements to create a new overview-oriented aspect. There is a strong correlation between dependencies and functionality which is often stated in the system requirements. This came up throughout the thesis, and it would be an interesting and worthwhile subject to try and get requirements into dependency management.
- Further investigate if different views of dependencies can be used for different purposes (similar idea as the 4+1 view model of architecture [1]). In dependency management, this would involve looking into what dependencies and functionality are important for different kinds of stakeholders in a company or project.

7 Conclusion

When developing a system with a modular architecture, the existence of dependencies is inevitable. During interviews, it was elicited that the presence of dependencies is often known due to the problems that they bring. Most often are they dealt with after the problems already have occurred. Some specific activities e.g. the build process of systems do manage their dependencies, but in general, dependencies are not handled nor taken into consideration for future development.

The first research question was to establish what dependency related problems that existed. Here the results show that companies are aware of several different problems at different stages of development, but a majority of them are not managed because of technical or practical limitations. Areas which were seen as most prioritized to solve were:

- Awareness and communication problems between developers or others within the development.
- Change impact and calculating cost-of-change for changes that spans over several components. Mainly to be able to prevent costly and complex quick fixes. This was said to be especially difficult when multiple teams were involved.
- Creating a way to represent and define dependencies and the be-

havioural structure of a system.

The second research question asked how to manage dependencies to tackle above mentioned problems. Analysis gave that it would be most beneficial to manage dependencies in a generalized way to easily gain an overview of the dependencies within a system. Managing dependencies cannot be too time-consuming for developers for it to be included in their daily work. The results suggest a definition of what a dependency is, how it could be documented and utilized to solve stated problems, its life-cycle and how it could be included in the development process. During validation, it was elicited that the aspect of dependency management is highly desired, but the biggest limitations of these results are recognized to be identifying dependencies and the difficulties of proving completeness.

Bibliography

[1] The 4+1 view model of architecture. *IEEE Software, Software, IEEE, IEEE Softw*, (6):42, 1995.

[2] Agile planning and development methods. *2011 3rd International Conference on Computer Research and Development, Computer Research and Development (ICCRD), 2011 3rd International Conference on*, page 488, 2011.

[3] *610.12-1990 IEEE Standard Glossary of Software Engineering Terminology.* Piscataway, NJ : IEEE / Institute of Electrical and Electronics Engineers Incorporated, n.d.

[4] Manoj K. Agarwal, Karen Appleby, Manish Gupta, Gautam Kar, Anindya Neogi, and Anca Sailer. *Problem Determination Using Dependency Graphs and Run-Time Behavior Models*, pages 171–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[5] S Agarwal and Agarwal P A. An empirical study of control dependency and data dependency for large software systems. *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference*, 2014.

[6] Aaron Brown, Gautam Kar, and Alexander Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 377–390. IEEE, 2001.

[7] Trosky B. Callo Arias, Pieter van der Spek, and Paris Avgeriou. A practice-driven systematic review of dependency analysis solutions. *Empirical Software Engineering*, 16(5):544–586, 2011.

[8] Zhenqiang Chen, Baowen Xu, and Jianjun Zhao. An overview of methods for dependence analysis of concurrent programs. *SIGPLAN Not.*, 37(8):45–52, August 2002.

[9] A Cockburn. Writing effective use-cases. *Humans and Technology*, 2001.

[10] Cleidson Ronald Botelho De Souza. *On the Relationship Between Software Dependencies and Coordination: Field Studies and Tool Support.* PhD thesis, Long Beach, CA, USA, 2005. AAI3200278.

[11] Peter Feiler. Configuration management models in commercial environment. Technical Report CMU/SEI-91-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1991.

[12] Philip WL Fong. Reading a computer science research paper. *ACM SIGCSE Bulletin*, 41(2):138–140, 2009.

[13] The Eclipse Foundation. Open source community website. `http://www.eclipse.org/`, 2017. Accessed: 2017-06-21.

[14] Vahid Garousi, Lionel C. Briand, and Yvan Labiche. Analysis and visualization of behavioral dependencies among distributed objects based on uml models. *Model Driven Engineering Languages Systems (9783540457725)*, page 365, 2006.

[15] Google. Google scholar search tips. `https://scholar.google.com/intl/en/scholar/help.html`, 2017. Accessed: 2017-06-26.

[16] A. Keller, U. Blumenthal, and G. Kar. Classification and computation of dependencies for distributed management. In *Proceedings ISCC 2000. Fifth IEEE Symposium on Computers and Communications*, pages 78–83, 2000.

[17] Bixin Li, Ying Zhou, Yancheng Wang, and Junhui Mo. Matrix-based component dependence representation and its applications in software quality assurance. *SIGPLAN Not.*, 40(11):29–36, November 2005.

[18] J. López-Martínez, R. Juárez-Ramírez, C. Huertas, S. Jiménez, and C. Guerra-García. Problems in the adoption of agile-scrum methodologies: A systematic literature review. In *2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 141–148, April 2016.

[19] R. L. Nord, R. Sangwan, J. Delange, P. Feiler, L. Thomas, and I. Ozkaya. Missed architectural dependencies: The elephant in the room. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 41–50, April 2016.

[20] P. N. G. Perera, A. R. D. C. Atapattu, H. T. Dias, N. T. Liyanage, R. O. C. Silva, P. S. Rupasingha, S. G. S. Fernando, and C. D. Manawadu. The impact of effective configuration management usage in software development firms in sri lanka. In *2013 8th International Conference on Computer Science Education*, pages 691–696, April 2013.

[21] A. Podgurski and L. A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Trans. Softw. Eng.*, 16(9):965–979, September 1990.

[22] A. Podgurski and L. A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Trans. Softw. Eng.*, 16(9):965–979, September 1990.

[23] Judith A. Stafford, Debra J Richardson, and Alexander L. Wolf. Architecture-level dependence analysis for software systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(4):431–452, August 2001.

[24] Constantine L.L Stevens W.P, Myers G.J. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.

[25] Ye Wu, Dai Pan, and Mei-Hwa Chen. Techniques for testing component-based software. In *Proceedings Seventh IEEE International Conference on Engineering of Complex Computer Systems*, pages 222–232, 2001.

Appendices

A
Interview Questions

The interviews started of with a short description of the goal of this thesis and about the interviewers (authors). The interviews were performed in a semi-structured manner and the following questions are created as a base to make sure that the interview keeps on track and that all areas are touched. Below each question is a short motivation of what information that is meant to be elicited from the question. Some motivations also include possible ideas to create follow-up questions if needed.

- What is your role in the development process?
  Context about who we're interviewing and to see is some problems are more significant to some roles than others.
- How many employees do you have in this company that is connected to software development?
  To gain knowledge if it's a large o small company. Makes it possible to see if there's a correlation between some problems and the size of the company.
- How many different roles do you have that take an active part of the development process?
  Architects, developers, testers etc. This is to find a possible correlation if some problems arise in separate groups of people that shall perform a specific task.

- How do you utilize components in your projects?
  Gain knowledge about of the company sees the area of components.

- How is the overall structure of the projects that utilize a component-based structure?
  Do they utilize Modules, microservices, hardware components, other components? Are these components from internal or external parties?

- What is the pros and cons by utilizing a component-based structure?
  What is the reason of why they utilize this structure? Are there some dependency related problems that they've noticed? Do some of these problems occur more often than others?

- How many different systems do you develop simultaneously?
  One or more is the core question. To find a possible correlation between some problems and multiple systems.

- How large are these systems and how many components are normally included?
  Deeper understanding of the structure of the systems. Is there a large difference between companies of how many components they utilize together?

- Do you have different teams working on different components?
  Structure of teams and how they are divided between the components. 1 team - 1 component? 1 team - many components? Gives a ground to if communication between teams is a possible problem

- When working with a specific component, are you aware of the existing dependencies of that component?
  - How did you gain knowledge about these dependencies?
  - What different kind of dependencies do you recognize?

  To what content are they aware of dependencies, examples of different dependencies that they are aware of and when in the development *process they gained this knowledge (when a problem occurred, design documents, other?).*

- How does the test process look?
  - How would you prefer it looked like?

  Different test-stages that is used. If only one component is changed do they test the entire system or parts of it?

- What different dependency-related problems have you encountered?
  - What problems do you rate as the most crucial or most hindering when developing software? Why?
  - What problems are hardest to find?

  The problems that arise with not managing dependencies. Examples of what the problem is and when it arise. A core question, continue to extract information about relevant and interesting points that is given.

- Do you know if a composition of modules works together before building the system?
  - Is this something you would like to do? Why/Why not?

  An idea of checking consistency within a component-based structure before building it with the help of dependencies. Is there a need of this functionality?

- Are there some components that is used in several different systems?
  - Is the component copied into the new system or is it the same component that is shared?
  - Has this brought up any new problems?

  Reusing components shall be one advantage with a modular architecture. What happens with the dependencies of a component when moved to another project? Is there a need of knowledge about its dependencies for it to be possible to reuse?

- Are there some entities regarding dependencies that is included into the version-control of the project?
  To what extent have they managed and documented dependencies or dependency related items?

- Is there any documentation made of existing dependencies?
  Design documents, In code, other?

- What tools and processes do you use to simplify development of a component-based structure?
  What existing tools and processes are used. Do some companies have a tool or process that eliminates problems.

- Do you believe knowledge about dependencies and including them in the CM-process would benefit development?
  Do they believe in what we're trying to do with this thesis.

- What CM-processes are practised when a change is about to be made to the system?
  To what extent is a change pre-analyzed. Change-requests, CCB (or other meetings), change impacts, cost-of-change calculations.