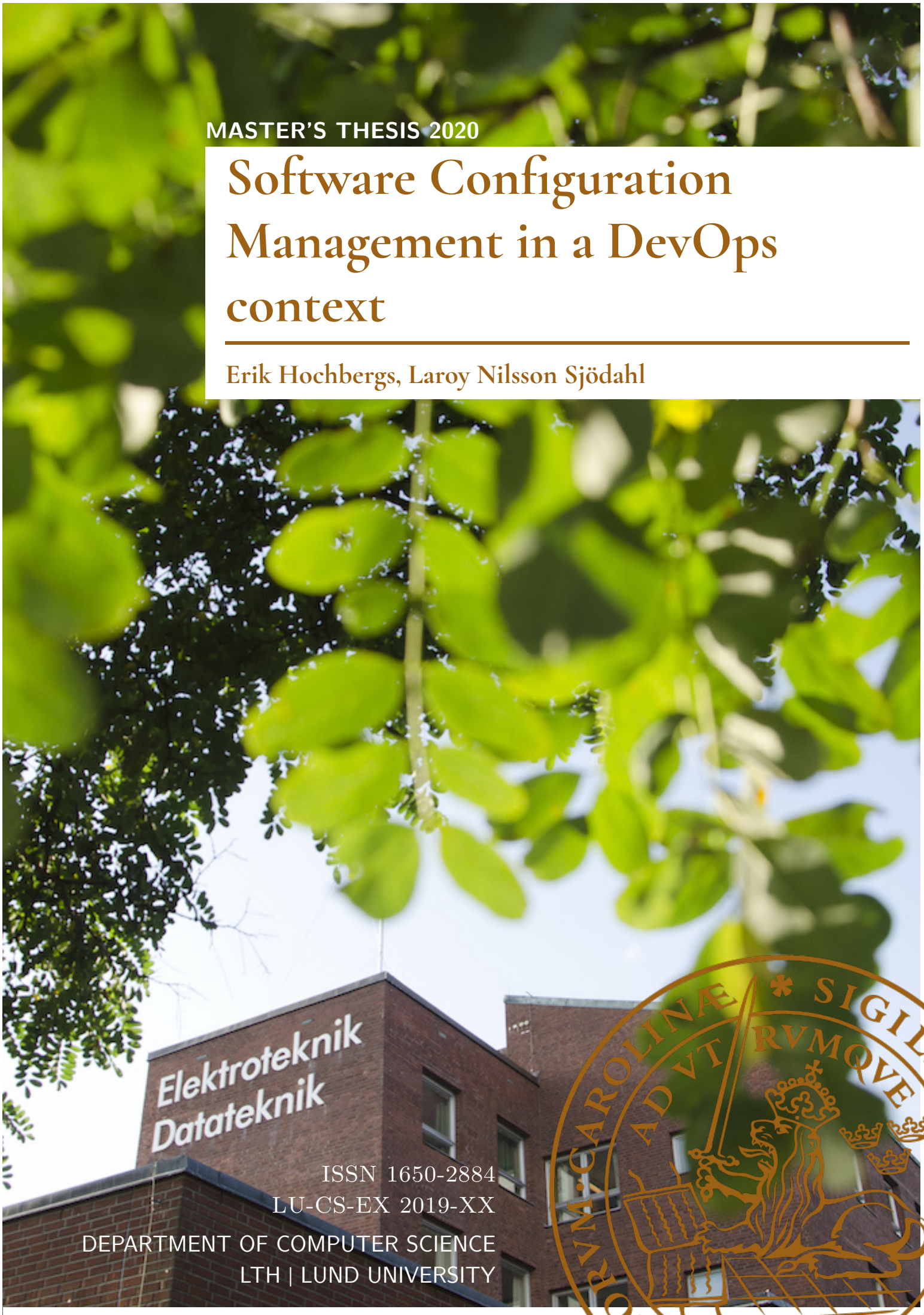

MASTER'S THESIS 2020

Software Configuration Management in a DevOps context

Erik Hochbergs, Laroy Nilsson Sjödahl

ISSN 1650-2884
LU-CS-EX 2019-XX

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-XX

Software Configuration Management in a DevOps context

Erik Hochbergs, Laroy Nilsson Sjödahl

Software Configuration Management in a DevOps context

Erik Hochbergs
tys12eh2@student.lu.se

Laroy Nilsson Sjödahl
fpr08lni@student.lu.se

January 29, 2020

Master's thesis work carried out at Praqma AB.

Supervisors: Christian Pendleton, cpe@praqma.net
Albert Rigo, alr@praqma.net
Lars Bendix, lars.bendix@cs.lth.se

Examiner: Ulf Asklund, ulf.asklund@cs.lth.se

Abstract

Software configuration management is rarely explicitly discussed within the context of DevOps, and vice versa. Literature on DevOps often mentions parts of software configuration management as being important, but specifics on how it is done within DevOps is often left out, which creates a disconnect between software configuration management and DevOps regarding its usage in DevOps, and the benefits of it. There is a lack of an operational definition of DevOps, which creates different expectations, because organizations and people have differing understanding on what DevOps is and how it is done, or can be done, in practice. DevOps has also become somewhat of a buzzword, which further obscures what DevOps is, and what one can expect from it. Depending on the source, there are different ways of accomplishing DevOps, but there appears to be principles and practices which are commonly used in DevOps. We attempt to map out these commonalities by characterizing DevOps based on literature and interviews with companies. We propose guidelines for adapting activities in software configuration management to a DevOps context. It is not intended to be a one-size-fits-all method, each part is treated as a piece which has a defined purpose, what problems it intends to solve, with examples of how it can be implemented.

Keywords: DevOps, Software configuration management, Characteristics of DevOps, Software configuration management activities, Internal dependencies, External dependencies, Guidelines, Graphs, Process plan

Acknowledgements

We would like to thank our academic supervisor Lars Bendix for his immense support throughout the thesis. We would also like to thank our industrial supervisors Christian Pendleton and Albert Rigo, and the rest of the office at Praqma, for their continuous support and feedback, and also for providing opportunities to participate in CoDe-Conf and Code Camp, which provided much inspiration and feedback.

Lastly, we want to thank the companies who participated in our interviews, and who also were part of our validation process, for their contributions.

Contents

Chapter 1
Introduction

Software configuration management (SCM) is a valuable tool for providing an infrastructure where items and information can move around, structured and controlled. It also provides support for collaborative activities, and touches on practically all aspects of product development (Bendix, 2019).

DevOps has become a very popular development method, and the term has become somewhat of a buzzword (Sharma, 2014). Because of its widespread usage no singular definition of DevOps exists, which creates different expectations based on different experiences and understandings of what DevOps is, and how it is done.

SCM, or parts of SCM, is sometimes mentioned in literature regarding DevOps as being important or part of DevOps, but specifics on its implementation or usage is often left out. In much the same way as with the DevOps literature, little literature seems to exist on SCM discussing DevOps, and its usage, which seems to point to a need for more research in this area.

This lack of a common definition, or operational definition, of DevOps, in addition to sparse information regarding SCM in this context, and the role of DevOps in SCM, presents difficulties in understanding the role of SCM in DevOps, why SCM can be valuable, or why it might not be, and how to use it. Because SCM is (or should be) method agnostic (Bendix, 2019), it should be possible to combine these two methods, which partly sets the premise for this thesis.

The purpose of this thesis is to explore what SCM activities, principles, and practices are needed in DevOps, and how these SCM activities can be implemented. In addition we provide guidelines for doing so within the context of DevOps, aimed towards both established companies and newer companies, e.g. start-ups, who wants to move towards DevOps and provide a source for how SCM can be adapted to it, and also how DevOps companies who wants to move towards SCM.

With this as background, our main research question was formulated:

- RQ1: *We want to explore and investigate what SCM is needed in a DevOps context and how to implement it.*

Which was further split in three parts as:

- RQ1a: *What SCM concepts and principles are needed and not needed in a DevOps context?*
- RQ1b: *What new SCM concepts and principles can be added to the SCM toolbox?*
- RQ1c: *How to adapt relevant SCM concepts and principles in a DevOps context?*

These research questions were explored in literature and through interviews, and resulted in a characterization of DevOps, providing a basis and context for our thesis on which we could analyze DevOps both through the literature, as well as how companies themselves viewed and implemented DevOps or wanted to implement DevOps, its relationship with SCM, also based on literature definitions and interviews with companies on how they used SCM, and lastly guidelines for how SCM can implemented in DevOps.

In the following chapter we will first provide background and context for the thesis, introduction to the problem, methodology, initial characterization of DevOps based on literature, and a short introduction to SCM, to provide the reader with the initial information needed for the following chapters.

Then follows the chapter Characterization of DevOps, containing the cumulative characterization of DevOps, based on both literature and interviews, as well as a graph showing connections between the identified characteristics.

Next, the chapter SCM in a DevOps context contains the cumulative definition of SCM, based on both literature and interviews, as well as a graph showing connections between different SCM activities. It also presents the resulting graph of combining the DevOps and SCM graphs, and subsequent analysis of this. Lastly, it provides the results in relation to our research questions.

Following is a chapter on discussion and related work, which presents a discussion of the thesis, related work, validation of the results, reflections and limitations of the thesis, and future work.

Lastly, a chapter concluding the thesis, containing a summary of the results of the thesis.

Chapter 2
Background

There is a lack of a unified definition of DevOps which creates different expectations, because organizations and people have differing understanding on what DevOps is and how it is done in practice. Since there are many opinions on how to do DevOps (and even whether DevOps is something you do, or if it is something you are (Kornilova, 2017)), DevOps is not necessarily the same process, or set of processes, in every development context, and misunderstandings occur when discussing the subject, due to different personal experiences, understanding, and views or opinions. Some compare DevOps to a buzzword (Sharma, 2014) or a feeling, rather than a set of concepts and principles, but also all of it at the same time. Even if the word "DevOps" has become heavily overused because of its popularity, an overall goal is still somewhat clear and we have tried to condense it to: The ability for organizations to efficiently deliver services and applications (TechInsights, 2013; Sharma, 2014). The details of how this is accomplished can differ, but there appears to be a subset of principles and practices that are commonly used in DevOps, which we attempt to map out. To accomplish the goal of DevOps, Software configuration management (SCM) needs to be taken into consideration. Regardless of development method used (e.g. agile or waterfall), certain parts are often present, e.g. a group of people with varying degrees of defined roles (e.g. a cross-functional team or a clearly defined role), who needs to work together to accomplish some goal. To this end, SCM is agnostic towards any specific development method. To accomplish this and to incorporate the overall goal of DevOps and keep up with the market speed, the workflow from idea to, for instance, a finished product, must be as efficient and effective as possible. In these cases SCM comes in very handy as it provides an infrastructure for a project or organization, which can cover almost any topic related to software development. What SCM very broadly does is identifying configuration items (CI) in a software product or project, controls these items, and reports and records change activities to the CIs. It does not matter if one work with testing, development, requirements, quality assurance, or project manager, SCM will be a great help. Moreover, SCM helps people to coordinate, which will in turn help with the workflow, when integrating each piece into an actual product. Therefore, knowing SCM and implementing the different techniques into projects will be a valuable asset (Bendix, 2019).

In literature on DevOps, SCM or components of SCM, are sometimes mentioned as a practice in DevOps, but the specifics on how it is done are left out (Jabbari et al., 2016), and any relationship or difference, or needs between DevOps and SCM can be difficult to see, and point out, without knowledge of them.

The thesis project was done with the help of Praqma. Praqma is an IT consultancy bureau that focuses on continuous delivery and DevOps. Their main goal is to help large and small companies deliver quick, safe, and sustainable software. This is a valuable asset when competing with the market demands. However, they do not only have a great knowledge in the philosophy of development, but also activities and tools surrounding it. Instead of being specialist around specific areas, they promote the wider knowledge, and see themselves as full stack developers. Therefore, Praqma became a great collaboration partner, with expertise in both DevOps and Software configuration management, which gave us valuable information and discussions during the thesis project.

Our master thesis will provide Praqma and others with guidelines of how to adapt SCM to a DevOps context. These guidelines will be the product of our result and conclusion of the thesis, which will in detail provide an analysis of these parts on how SCM can be done in DevOps. This in turn will make companies more aware of the importance of SCM in DevOps, and what SCM to use in DevOps, and help them along the way to use and improve DevOps in their organization.

In the coming sections the problem will be discussed, and the research questions presented in the context of the problem. The chosen methodology for the thesis project will be presented, and certain choices regarding it will be discussed. An initial characterization of DevOps will be presented, and lastly a short introduction to Software Configuration Management which presents some of the most important topics for understanding the rest of the report.

2.1 Problem statement

In this section the research questions will be formulated based on the background, and the discussed problems, will be listed, and motivation and discussion will be provided for each.

Analyzing the outlined background and problems, and the lack of resources explicitly discussing SCM and DevOps, there appears to be a need to explore the relationship between them, and how SCM can support and be a part of DevOps, as well as to provide a resource which discusses this clearly, using relevant terminology. This led to the formulation of our overall research question and goal:

RQ1: *We want to explore and investigate what SCM is needed in a DevOps context and how to implement it.*

From RQ1, three parts were identified as additional research questions, by separating it, to help answer RQ1 in its entirety with higher granularity. Because it is unclear what SCM is needed in DevOps, that is the first part to explore to get a clearer image of the SCM needs in DevOps:

RQ1a: *What SCM concepts and principles are needed and not needed in a DevOps context?*

While investigating this, we also want to explore the possibility of the addition of concepts or principles not necessarily covered in SCM already, but could be, which might have a positive impact, or is perhaps already used in DevOps:

RQ1b: *What new SCM concepts and principles can be added to the SCM toolbox?*

Finally we want to investigate how these concepts and principles both can be implemented, and also how they might be implemented already, in a DevOps context:

RQ1c: *How to adapt relevant SCM concepts and principles in a DevOps context?*

In order to investigate these research questions, characteristics of DevOps needs to be established, for us to be able to analyse SCM within these parts that are part of DevOps. This will be done in two parts, literature study to establish a characteristic based on theory, both academic and other published resources. The second part will be through interviews with companies, who are self-identifying as DevOps, or are moving in that direction, to collect their views on DevOps, how they work, and their views on our characteristics. This will then be analysed and compiled into a set of characteristics which the research questions will focus on. We decided to not formulate this part as a research question, as it is not our main focus to provide a definitive characterization or definition of DevOps, but it is needed to build a basis for the research questions within the context of DevOps in this report.

2.2 Methodology

This section discusses and analyzes how the thesis was conducted. Given that our research questions are closely connected to DevOps as a basis, the initial step will be to characterize, through literature study, DevOps as a process and its overall problems and needs in order to have a well defined-context for the three sub-parts of our research question. This will provide us with a number of characteristics outlining DevOps, not defining it, which we can use in our later analysis. We also compile a list of important topics in SCM through literature study in parallel.

After this initial step we wanted to collect information from companies who produces software, and who self-identify as DevOps, to map their processes to our prior analysis, to further strengthen our classification of the characteristics of DevOps, and provide context to their specific situation. The reason for this initial step is that an initial cursory search for a description or definition of DevOps proved difficult, and an example is the wikipedia page on DevOps, which is severely lacking for our purposes.

Because of time constraints, the number of interviews has to be limited, but enough to provide information. Therefore, four companies was interviewed with a focus on DevOps, as the characterization and classification of DevOps was not our main focus, followed by six interviews with companies regarding SCM in their DevOps. More interviews for the SCM part, because this is our main focus. This was followed by an analysis of how SCM is used in companies or teams, which self-identify as DevOps, to identify the relationship between them, if there was one, which in turn would help us answer our research questions. This anal-

ysis was performed iteratively, as data was collected during the course of the project, and was the foundation of formulating guidelines for using SCM in DevOps. These guidelines were send out to eight companies (all the interviewed companies plus one other) for validation purposes, this to ensure the sufficiency of the guidelines and letting external parties provide feedback regarding the validity of our results.

2.2.1 Information collection

There are many ways to collect data, such as questionnaires or surveys, focus groups, case studies, and interviews. When deciding which methods are appropriate, we have a need to balance the width and depth of the data collected, as well as the control of precise feedback while and after collecting information. This section contains the information collection activities done during the phase before the analysis part of our thesis.

Literature study

Our lack of knowledge, and the poor description and definition of DevOps on pages like wikipedia, made the literature study necessary. This to make the master thesis trustworthy, and not take place in a vacuum, without any references. Therefore, to find and collect reliable literature we first had a brainstorming session, where we found a best fit process for us. The process started with a collection of literature, where we both read abstracts and skimmed through papers to determine relevance. The literature was collected with the help of our supervisor Lars Bendix, and the platforms Lubsearch and Google Scholar. When this was done, we did a more thorough reading on the literature that was collected. During the reading we wrote down pieces that we reacted on, and thought would be useful, followed by a frequency analysis on the data. This was done to find words that was frequently used in the sources describing principles, practices, or other noteworthy topics. Lastly, the resulting list of terms collected were the basis for our initial set of characteristics of DevOps.

Interviews

We have chosen to mainly focus on interviews. The individual interviews will have a semi-structured approach. This means there are going to be questions that have a more of a direct answer and questions that are more open for discussion. With the participant's permission each interview will be recorded, to allow for both of us to take a more active role during the interviews, instead of focusing on parsing the conversation as it happens. This also allows us to review the interviews in greater detail afterwards.

The choice of interviews is based on wanting to have flexibility, the ability to follow up immediately on a given answer or thought in a discussion, and to be able to steer the conversation based on the answers. This is much more difficult and time consuming if other forms of information collection is used, such as questionnaires or surveys (since you would need to follow up afterwards). There is also the ability to control the depth of each interview as deemed needed, while the width is controlled by the number of interviews in relation to the number of companies.

When constructing the interview questions, the characteristics identified during the literature review formed a basis for what information the questions should generate. A brain-

storming session around these characteristics formed a number of broad questions regarding a software development process, with the purpose of steering the conversation towards these subjects. From each broad question a number of follow-up questions were also brainstormed, with the purpose of collecting specific information on the subject, if the interview participant did not touch on it. These questions were pilot tested by employees at Praqma to verify that the questions are valid in our context, and generated the wanted information. After each performed interview, the questions were also evaluated according to what worked in practice, and what did not. The changes made to questions based on this feedback were e.g. specific wordings, and additional follow-up questions, addition of entirely new questions, or removal of them. In order to get a better view of the participating companies, and their representatives, we have collected a short summary, describing, e.g., the type of company, the role of the interviewee, and which interviews they participated in. This can be viewed in the matrix in 2.1.

Company	Type of company	Size	The interviewer's role in the company	Kind of interview
Company 1	Agile	Established	Architect consultant, where the main role is how to do DevOps in the company	DevOps
Company 2	Parts are waterfall, development moving towards agile/DevOps	Established	35 years at the company, 15 years Configuration manager	DevOps & SCM
Company 3	Company that works with both software and hardware (applications and embedded systems), moving towards agile/DevOps	Established	Official roll is Configuration manager, helps the company to develop tool chains and to adapt to Devops	DevOps & SCM
Company 4	Agile	Established	Configuration manager, six years as a consultant at the company, where the main role is to help Development teams with DevOps and IaC	DevOps & SCM
Company 5	Parts are waterfall, moving towards agile/DevOps	Established	System engineer	SCM
Company 6	Agile software company	Established	Worked as a configuration manager for 15 years, but is now a build manager at the company	SCM
Company 7	Parts are waterfall, moving towards agile/DevOps	Established	Product Handling Principal Developer and Configuration manager	SCM

Table 2.1: Description of the participating companies interviewed

2.2.2 Analysis

To provide a context for the data and extract useful information, we had to understand, interpret, structure, and explore the collected data in meaningful ways using analytical processes. This section contains the analysis process for DevOps and SCM, which led to Figure 4.2, where connections between SCM activities, DevOps characteristics, and DevOps activities were drawn.

Analysis of collected DevOps data

For the interviews on characteristics of DevOps, each recorded interview was summarized in a bullet list shortly after each interview, sorted under the general theme of the question being discussed. The interviews were divided among the two of us for efficiency when more than one interview was held in a short time period (otherwise both would analyze and compare), and certain parts we found difficult to interpret were analyzed by both, and the resulting bullet lists were also inspected by each of us.

This allowed us to quickly get an overview of what subjects related to our characteristics from the literature study each interview touched upon, and also details regarding them for later discussion. The interview lists was then broken up, and with the use of an affinity diagram (Arvola 2014) each item were mapped to one of our DevOps characteristics, to organize the collected data, which allowed for a quick overview of the most commonly discussed or used characteristics based on the interview data.

During the analysis of the interviews and the mapping of characteristics, we did further literature studies. The data from the literature studies, and the data from the interviews bullet lists were then used to strengthen our classification of characteristics of DevOps, and to produce a compiled result. We used the compiled results to rank the DevOps characteristics after importance. However, in order to add more value to the research and to further analyze the DevOps characteristics a directed graph was made, where the directed graph was used to map dependencies between the different characteristics.

Analysis of collected SCM data

As in the analysis of the DevOps data, each interview of the SCM activities was recorded and summarized into a bullet list shortly after each interview, sorted under the general theme of the question being discussed. However, the SCM interviews were more scattered than the DevOps interviews, which led to us both having time to analyze each interview, and inspect each resulting bullet list. To organize the collected data, and provide a quick overview of the most commonly discussed SCM activities an affinity diagram were used, where each item from the SCM interview was mapped to an SCM activity.

Through our interviews, we also analyzed how SCM could be used in practice, and investigated differences (if any) between the literature study and the interviews responses to strengthen our classification of SCM, and in turn produce a compiled result. The compiled result was then later used to draw a SCM graph, where the graph clarified and visualized the connections and dependencies between the SCM activities, which created a better understanding over the internal sequences.

2.2.3 Alternate methods

While deciding on appropriate methods of collecting information, we need to account for the time constraints of the thesis project, which restricts us from using certain methods to the extent that we would like.

Case studies provide great depth, but are time-consuming, which might limit the number of companies studied greatly, which potentially goes against our initial goal of providing a wider solution. Questionnaires are a simple way of collecting more data quickly, but their

generally static format and the fact that people might very well interpret questions differently might provide more uncertainty in our data. The lack of ability to immediately follow up on interesting parts, or clarification and expansion, is a clear negative. Focus groups provides great range and depth, but can be harder to analyze and to organize.

2.2.4 Validation

During the course of the thesis project, Praqma provided the opportunity to hold a workshop regarding the characteristics of DevOps. The workshop was designed to focus on discussion between the participants, and for us to answer any questions or clarify when needed. The time at which the workshop was held was very close to the preliminary completion of our subtask of characterizing DevOps, which provided a timely opportunity to get feedback on the results and to validate them. Specifically, the workshop was focused on our graph describing dependencies between DevOps characteristics, and discussions on both the characteristics themselves, and the dependencies. We also submitted this report to a number of companies for external validation, of which the results are presented and discussed in chapter 5.

2.3 Characterizing DevOps

The lack of a unified definition of DevOps presents an obvious problem, mainly that in order to perform interviews on the subject, and to investigate our research questions we want to have a basis for discussion, and concrete information that characterizes DevOps. This section describes common characteristics of DevOps (table 2.2), as identified during our literature study, and provides an academic characterization of DevOps, which we use as a basis for educating ourselves, and in extension for preparing interviews and subsequent analysis and refinement of these characteristics in combination with what we encountered in the industry.

Holistic

The DevOps culture want, among other things, to increase trust, respect and understanding (França et al., 2016; Ståhl et al., 2017). Therefore, in DevOps it's important to have a holistic view. For instance, this means that it doesn't matter if ones code is in QA, the production server, or in development. You as a team are responsible for your code in all environments, and you have to make sure that it performs as expected (Casey, 2019). Moreover, a holistic view is key to increase the understanding and decrease the gap between business stakeholders (Sharma, 2014).

Culture/Social aspects

The "social aspects" and "culture" of DevOps is mentioned frequently in literature, which probably is an indication of value (França et al., 2016; Sharma, 2014; Ståhl et al., 2017; Bendix and Pendleton, 2019; Erich et al., 2014; Kornilova, 2017). As well as describing it as a DevOps culture, which should be encouraged, to promote DevOps principles and practices. Examples mentioned are trust, communication, being open to change, respect, personal responsibility, cultural aspects, collaboration (França et al., 2016; Ståhl et al., 2017). Others describe it more

concisely as a focus on people, processes, and tools, to promote the DevOps culture (Sharma, 2014; Bendix and Pendleton, 2019).

Automation

Automation refers to automate tasks where possible. This includes any task that is repetitive and performed manually frequently (França et al., 2016). Reasons for this are that manual, repetitive tasks consume time and effort, may lower consistency, and repeatability, so automation counters these issues and provides reliable and repeatable processes (França et al., 2016; Sharma, 2014).

Quality assurance

Quality assurance is broadly a way of assuring quality of products or processes, e.g. by following an established standard. The importance of quality assurance in DevOps is emphasized in many sources (e.g. (Kornilova, 2017; França et al., 2016; Jabbari et al., 2016)). Using quality assurance throughout the implementation of DevOps, the quality of the involved processes and products can be monitored and evaluated (França et al., 2016; Roche, 2013).

Shift left

Shift left moves testing and QA, operations earlier in the delivery cycle. This means that, for instance, the quality for a product will increase and test cycles will be shortened. Moreover, by moving operations concerns earlier in the process, the development and quality assurance can test against systems that behaves like the product system. This means that in DevOps one can see how the application performs and behaves before deployment, which, for example, reduces unpleasant surprises in production (Sharma, 2014). Furthermore, with shift left down time and security issues will decrease, which will make the company to move faster (Kornilova, 2017).

Teamwork/Collaboration/Communication

To get an efficient teamwork between operators and developers, there needs to be a good set of collaboration and communication among the teams (Jabbari et al., 2016). Collaboration and communication is therefore key for DevOps to work. However, keep in mind that these two practices does not solve all problems. For instance, good communication and collaboration, but slow processes does not lead to better deployments (Sharma, 2014).

Monitor/Measure

Often mentioned as a basic component of DevOps as an important part of operations, monitoring and measuring is intended to be incorporated early in the development cycle, to not only monitor a system in production and customer feedback, but also the functional and non-functional aspects of a system in development (with e.g. continuous automated testing) (França et al., 2016; Sharma, 2014). The purpose is to monitor characteristics of an application or system, and collect metrics, which acts as feedback to the application or system, different processes in e.g. development or production, and provides an overview of a product

or system in development and in production based on these metrics in a way that is accessible and understandable to the people involved. These can then be used to create new ideas, improve processes, discover problems, used in planning, validate quality, etc. (França et al., 2016; Sharma, 2014; Erich et al., 2014; Jabbari et al., 2016; Kornilova, 2017; Ståhl et al., 2017).

Sharing

Sharing refers to spreading and sharing knowledge and information to team members and relevant parties (e.g. (Erich et al., 2014)). This includes for example project information and information about principles and practices regarding the development process in the context of DevOps, expertise from specific people, and making them accessible by everyone involved in order to lessen dependencies on certain people and provide information about the project, and create an environment where team members are educated, informed, and understand their processes and goals. (França et al., 2016).

Feedback loops

In DevOps it is important to make organizations react, and make changes, more rapidly. To achieve this the organization needs to get quick feedback, which are based on e.g. metrics collected and tracked during the monitoring activities, or customer feedback, and which are then acted upon by e.g. adjusting project plans and priorities, production environments, changing release plans. This is done by amplifying feedback loops (Sharma, 2014; Auerbach, 2017). To amplify these feedback loops there needs to be communication channels that allows stakeholders to act on and access all feedback (Jabbari et al., 2016; Sharma, 2014).

Integration

As mentioned over and over in the literature, continuous integration is a big part of DevOps (Kornilova, 2017; Sharma, 2014). Furthermore, when doing frequent integration one decrease the integration cost because changes are detected and resolved early in the development life cycle (Bendix and Ekman (2007)).

Delivery

The key point of DevOps is to reduce the cycle time for a service or product. This includes everything from initial idea to production release to customer feedback to improvement on that specific feedback. (Sharma, 2014; Jabbari et al., 2016). To achieve a reduced cycle time one can, for example, automate the delivery pipeline, react quickly to changes, add continuous feedback, add continuous integration and shorten release cycles (Jabbari et al., 2016).

Testing

Testing, including continuous (and especially automated) testing, is an important part of DevOps, which is used continuously during the development process to e.g. assure quality, correctness, get fast feedback (Daskalopoulos, 2019; Sharma, 2014; Jabbari et al., 2016). It is also recommended to employ automated testing as part of the tracked metrics in the monitoring activities (Sharma, 2014), and that testing is performed in an environment similar to

the production environment, which helps keep the product in a "shippable" state (Ståhl et al., 2017).

Planning

Another principle that has to be included in DevOps is planning (Sharma, 2014; Jabbari et al., 2016). A release plan offer capabilities to customers, which can include, among other things, project plans, release roadmaps and delivery schedules. These release plans often includes meetings and spreadsheets, which stakeholders across the business take part of (take time). However, explicit processes and automation enables predictable releases, and eliminates the needs of spreadsheets and meetings (Sharma, 2014).

Deployment

Deployment is the set of actions that makes a product, or update, available for use. In DevOps one goal is to optimize this process, and key factors that are often mentioned are continuous integration and continuous delivery, that aims to automate the process of getting the product deployable, and in extension the deploying itself, continuous deployment (Sharma, 2014; França et al., 2016; Auerbach, 2017; Jabbari et al., 2016).

Change management

This title is more of an umbrella term used for cases where handling changes is discussed. Few sources explicitly discuss change management in the sense of structured approach, as in SCM (but some mention it as a practice in DevOps, e.g. (Jabbari et al., 2016; Auerbach, 2017) which also mentions change advisory board meetings, which is part of the change control process in ITIL). Several sources stress the importance of being able to react to change, e.g. (Jabbari et al., 2016; Ståhl et al., 2017), or reacting to change over following a plan (Ståhl et al. 2017; Casey 2019) and focusing on smaller releases (Jabbari et al., 2016). Change is also mentioned as a metric that can be tracked for QA purposes, e.g. change fail rate, deployment rate, and time to deployment (Kornilova, 2017), and also tracking changes in general, for e.g. traceability between code and deployment (Auerbach, 2017).

Infrastructure as code

The concept of infrastructure as code is often mentioned together with DevOps (Jabbari et al., 2016; Kornilova, 2017; Ebert et al., 2016). It refers to treating the infrastructure (such as networks and virtual machines) as code, so the configurations can be version controlled and tested, and debugged, using the same tools the DevOps uses for the source code, and provides a consistent environment which helps reduce issues that might appear otherwise, such as bugs because of different configurations (Ebert et al., 2016; Guckenheimer, 2017).

Holistic	Culture/Social aspects	Automation	Quality assurance
Shift left	Teamwork/Collaboration	Monitor/Measure	Sharing
Feedback loops	Integration	Delivery	Testing
Planning	Deployment	Change Management	Infrastructure as code

Table 2.2: An overview of the characteristics of DevOps

2.4 Introduction to SCM

Software Configuration Management (SCM) is a task that involves identifying, organizing and controlling changes and modifications to software, in order to e.g. establish what components make up a system, different versions or variants of a system, how to handle changes made to these components, making the status of these components visible, and enabling collaboration through coordination of work with principles and practices, processes, and tool support. Because this thesis is mainly focused on SCM, it is important to have some understanding of what it is, and be familiar with important areas of it. This section will list and describe topics in SCM (table 2.3) that we will later use in our thesis as a basis for analysis and discussion.

Configuration Identification

Configuration identification refers to identifying distinct components of a given system to be managed by software configuration management. These are called configuration items, and their features are recorded and versioned separately, which adds traceability during a project where each issue or change can be mapped to a unique configuration item (Bendix and Ekman, 2007). The configuration items can be a single item, or a collection of many, e.g. a module containing multiple files. The granularity of the configuration items identified during configuration identification should be decided by the value it adds (Bendix and Ekman, 2007; Kelly, 1996).

Configuration Control

Configuration control formalizes changes in e.g. software, through a change control process. This process includes evaluation of the change request, coordination, approval of the change request (or disapproval), and lastly the implementation of the proposed change, followed by verification of the implementation. A change control board (CCB) performs the evaluation, and an impact analysis to evaluate the impact the change would have, and is then implemented by a developer, and is tested to verify the implementation is correct according to the change request (Bendix and Ekman, 2007; C&A, nd).

Configuration Status Accounting

Configuration Status Accounting provides the ability to get metrics about configuration items (CIs) and activities related to these. This could for instance be progress reports, transaction logs, and change logs, CCB actions, or feedback from impact analysis. With Configuration Status Accounting one can at any time, during the life cycle, track information about

a product, which in turn provides traceability for the system, its components (the CIs), and decisions. The CSA also provides visibility for a system by organizing and tracking these changes and decisions during a project, and presents a full view of the system and its components, and their status, at any given time (Bendix and Ekman, 2007; Daniels, 1985). The persons who are authorized to gather the information could be anyone in a project - tester, programmers, QA to mention some. Furthermore, a modern Configuration Status Accounting uses a Configuration Management Database (CMDB), where users send, so called queries, to get real-time data (Bendix, 2019).

Configuration Audit

The purpose with audits is to evaluate the product, and see if it meets the configuration documentation and if all change requests are met. One can carry out an audit when the product has reached a sufficient level, which could be, for instance, when approaching a release date. There are two different kinds of formal audits; the physical configuration audits and the functional configuration audits. The difference between these formal audits are that the physical configuration audit focus on, if the product meets the physical description of the product, while the functional configuration audit is a type of check to see if the product is performing according to changes, specifications and requirements (Daniels, 1985). The team that guarantees that the SCM activities has been controlled and applied during a in-process audit are often the QA team (Bendix and Ekman, 2007).

The outcome of a configuration Audit doesn't need to be perfect. In the end it's up to the project manager to decide if the product should be released or not. However, when a Configuration Audit gets approved a new baseline for the product will be created. The baseline creates a stable product version that will be remained untouched until the next Configuration Audit (Bendix, 2019).

Version Control/Collaboration tools

Version control tools is a valuable asset to keep track of history for configuration items (documentation or source code). The version control tools provides the ability to store configuration items and set versions. The versions that are set can not change, and are unique, which means that at any time in the process recreation of a version can take place. The items being stored are accessible at all times, and can be shared along project members, which provides automatic support for Configuration Status Accounting (Bendix and Ekman, 2007; Asklund et al., 2004). When sharing files along project members, each member should isolate themselves, and create virtual workspaces. This to prevent members from changing files directly in the repositories. However, this in turn results in two new problems; simultaneous updates, and double maintenance. Simultaneous updates occur when members make e.g code changes to the same version. This creates the risk, when changes being uploaded, that one overwrites the other. Double maintenance occurs when one isolate themselves within their own workspace. This lead to that multiple workspaces will be created. These workspaces will, after some time, cease to be identical, and several versions of each file existing simultaneously, and needs therefore to be updated with any changes made during the time of isolation before being able to update the repository, and keep oneself synchronized with the repository being used. To solve these three problems, which were introduced by Wayne Babich, one needs to

have a good set of collaboration tools (Bendix and Ekman, 2007).

Build Management

Build management refers to the process that handles the building and defining of a system. When building a system, the many components, modules, and dependencies are collected in a system model, from which all these parts are linked together. A given system model can be used to derive different variants of the system using a build management tool, and this can all be automated (Bendix and Ekman, 2007).

Teamwork/Parallel work

Teamwork includes effort, interaction and communication, which needs to be effectively executed and organized for it to work. There are three important issues when working in parallel;

- Coordination: Everyone needs to understand what colleagues do, and why. For instance, there needs to be frequent communication about baselines and changes to the entire team, and stakeholders.
- Roles and responsibility: Everyone being responsible often leads to no one assuming responsibility. There needs to be people that can be held accountable for milestones, changes on components, and features outcomes.
- Complexity: Complex tasks needs to be broken down, and distributed to owners with suitable milestones, which can in turn be tracked and monitored to measure and communicated to team members (Appleton et al., 1998).

In order to collaborate in the team, and to efficiently perform work in parallel, a well planned development environment is needed, which includes collaborative tools such as version control tools, which lets you work in parallel, in accordance with the previously mentioned benefits of version control and collaborative tools.

Workspace Management

The version control tool keeps track of different configuration item (CI) of different versions in a project, which are often stored in a repository. In the repository, through the version control tool, the history of each CI is available, and a given workspace can be composed of any configuration of these versions, not only the latest version. It should not be possible for a developer to work directly from the repository. This is because the versions that are kept in the repository should be immutable. To be able to make changes one needs to make a copy of the chosen documents or modules, make the changes, and then add the change to the repository. This in turn will make it possible for developers to work within a controlled environment, which means that they will not be affected by changes from other developers. Workspace management creates the ability to make a workspace from a set of files from a repository, make changes to these files, and to add the changed files to the repository. However, when making changes to files in the workspace, the workspace management must also provide the ability to update the workspace with selected files from the repository. Problems

avoided using a personal workspace are e.g. working on the same files (shared data problem), and double maintenance by updating the repository from each isolated workspace. A good version control tool will also make sure you apply your changes to the latest version of the file(s) before being allowed to update the repository (simultaneous update problem), which involves updating your workspace first, in case the version in the repository is no longer the same as the version your own files are based on (the version of the file(s) you originally checked out). (Bendix and Ekman, 2007; Asklund et al., 2004; Feiler, 1991).

Change Management

It is important that change management cover all changes to a system. Change management needs to track changes from the origin of the change to the implemented source code. To be able to do this, change management needs to include processes and tools that support the tracking of changes and the organization. Also, there needs to be some sort of traceability between changes and its implementation. Therefore, these tools needs to be tools that gather explicit data under the process of making a change request, and the whole process until eventual implementation and preserves traceability between the change request and its corresponding implementation. Another aspect of change management is that it also provides important metrics about a project (Bendix and Ekman, 2007).

Release Management

Release management handles formal releases to a customer, and also internal and less formal releases, e.g. a developer releasing to a project. A formal release needs physical and functional audits before being released, to either establish a baseline for the system, or verify the baseline against e.g. documentation so that the correct system has been built, or both. Releases to a project can be a developer applying changes in code, and how and when this integration of changes occurs needs to be established. A bill of materials might be used to record how the release was built, and what it consists of, to ensure that the release can be recreated. Many of the steps of a release can be automated, e.g. test cases, integrating the changes into the code, delivery of the changes to an environment, and even deploying to a customer, and part of release management is deciding how and when (and if) each of these things are to be performed (Bendix and Ekman, 2007).

SCM Plan

A document describing the SCM activities regarding an organization or a project. The SCM plan acts as a system, or reference, to be used, and creates awareness about the SCM activities amongst the involved parties. The SCM plan can be implemented incrementally, but the overall outline of the “final” SCM system (not necessarily all the details) could be specified from the beginning, and then introduced in steps (Leon, 2014).

Configuration Identification	Configuration Control	Configuration Status Accounting
Configuration Audit	Version control	Build Management
Teamwork/parallel work	Workspace Management	Change Management
Release Management	SCM Plan	

Table 2.3: An overview of the described SCM topics

Chapter 3
Characteristics of DevOps

This chapter provides the compiled results of our characterization of DevOps through our initial literature study, literature encountered during the DevOps interviews, and the following interviews performed concerning DevOps in practice at four companies who uses or are moving towards DevOps. This will in turn strengthen our classification in section 2.3 of the characteristics of DevOps in the cases where there are overlaps, and also provide differences, e.g. between different types of companies and how they might implement these characteristics. Differences and changes from the initial characterization through literature study will be discussed under each of the listed characteristics, and this chapter will represent the characterization of DevOps we use during the rest of this report.

Culture

In interviews, speaking to different industry people, and during the DevOps conference "code-conf 2019", issues regarding the cultural, social, and holistic aspects were usually discussed under the umbrella term Culture. In accordance with this, we have also combined these characteristics under the term Culture, and refer to section 2.3. Most interviews emphasized responsibility, both as a team and on a personal level. The responsibility extends outside the produced code, and encompasses the entire development process, including e.g. pipelines. Also removing dependencies on specific people who might have specific and critical knowledge, since that essentially creates an internal handover within the team, and isolates knowledge instead spreading it.

Culture itself is self-reinforcing (Fournier, 2019), which means that the culture should probably extend outside teams themselves, or at least not having conflicting principles and practices on the outside.

It has also been suggested that a culture focusing on optimizing the sharing of information, trust, innovation, and risk taking, is a good metric for performance (Forsgren et al., 2019). In order to foster a culture encompassing these types of characteristics, it is important to also maintain a measure of psychological safety within the culture. This means that team

members should feel safe to take risks, try new things, and make mistakes, and be able to be vulnerable and be able to ask questions in a friendly environment (Fournier, 2019; Forsgren et al., 2019). With this wide scope, culture definitely appears to be a very important aspect of DevOps as a whole.

Automation

The interview answers regarding the characteristic automation correlated well to section 2.3. They all agree upon that automation should take place where possible. Some focus areas for automation from the interviews are:

- Testing, to better support continuous testing,
- Logging for monitoring purposes,
- CI/CD and deployment pipelines,
- Infrastructure.

This is to produce reliable and repeatable processes, by decreasing manual and repetitive tasks, which in turn decreases errors in a system (Smeds et al., 2015). The automation characteristics also support DevOps-capabilities such as planning, collaboration, integration, testing, monitoring, feedback, and other characteristics throughout development, which provides a more stable service and software development process. This in turn help employees to be more effective and productive (Smeds et al., 2015). Moreover, automation is required in the operation processes in DevOps. This to keep up with the market speed, make the operation processes more flexible, and eliminate manual processes. Furthermore, to improve and ensure the quality of products test automation is a necessity in the development process (Lwakatare et al., 2015). With this said, we can with confidence say that automation is an important characteristic of DevOps.

Quality Assurance

Assuring quality (chapter 2.3) was important during the interviews, and there were some differences in the implementation of this. In some cases there were established Quality Assurance teams (for example in cases where the companies themselves and their products were more strictly regulated, and often when the company also produced hardware), and in other cases, the development processes themselves represented a degree of assurance in terms of quality, e.g. by continuously testing, monitoring, and usage of an automated pipeline. There seems to be an emphasis on QA being part of DevOps, rather than an external group (as that would add a hand-over which breaks the flow of DevOps, and e.g. interrupts a CI/CD environment), and through other aspects of DevOps achieve QA, e.g. an automated CI/CD pipeline likely has some form of testing which is run already (Yigal, 2016; Lwakatare et al., 2015). Based on this, we still value QA as a concept highly, and with the reasoning that other processes in DevOps directly, or indirectly, contributes to maintaining and improving quality, and assuring quality throughout development and beyond, the sum total of QA can likely be very high without the need for a dedicated QA team.

Shift left

Not explicitly mentioned during interviews, but the general meaning was present in most, as in chapter 2.3. Many wanted to introduce testing and monitoring early in the development cycle. For instance, by moving continuous testing, monitoring and continuous integration sooner in the process, we can prevent, detect and correct errors earlier in the process. This in turn is less time consuming than waiting for a formal review (Forsgren et al., 2019). This was confirmed in the interviews, where they all mentioned that they, after introducing testing and monitoring early, were able to find defects and errors earlier and faster in the delivery lifecycle. Something that was not mentioned in the interviews, but was found in the literature, was to let operation work side by side with development. Failures that is observed in production is rarely encountered early in the development life cycle. This because, the procedures for development and operations may differ. Operation procedures can be much more manual, and use different tools than the development procedures. Letting operation and development work together will reduce the failure rate in production (Schrag, 2016). As mentioned in section 2.3, another way to lower the failure rate in production is to make all environments look more like production. By reducing the failure rate in production the cost for application failure in production will most likely decrease (Schrag, 2016). Therefore, shift left can be seen as an important characteristic of DevOps.

Collaboration

As mentioned in all interviews, collaboration is an important characteristic of DevOps. How they performed this collaboration and communication could vary from company to company, and team to team. Some companies had distributed teams, which often led to communication and collaboration through chat platforms, discussions forums, and formal meetings. However, if the teams were close, Ad-hoc, informal meetings, open doors (meetings with physical contact) was more common. DevOps tries to extend the collaboration between development and operational personnel. By doing this, there will be a higher degree of information sharing, a broadening of skill sets, and an increased feeling of shared responsibility between the two organizations (Lwakatare et al., 2015). Furthermore, by introducing more cross-team collaboration there will be an increased confidence in security posture (Mann, 2019). With this wider knowledge, we can say that collaboration is an important aspect when working with DevOps.

Monitor/Measure

Monitoring was a DevOps characteristic that was mentioned in all interviews. Accordance with section 2.3, monitoring was integrated early in the development cycle. Both development and operation could take part of the monitored data through dashboards. The dashboards could, for instance, present customer feedback in production, and functional and non-functional aspects in development. An organization that follows the DevOps approach should have the capability to detect and recover from service failures without delays. To detect these service failures immediately, it is a necessity to have monitoring (Smeds et al., 2015). Another challenge with monitoring in DevOps is the process of monitoring relevant data in an effective way, which can be solved by introducing the DevOps characteristic collaboration between development and operation (Lwakatare et al., 2015).

Sharing

Mentioned in all interviews in varying degrees in accordance with section 2.3, and efforts to share knowledge were present on some level everywhere. Some experienced difficulties with certain key people who maintain their fixed role, which introduces handovers within teams, which isolates knowledge, and reduces the workflow.

Some specific measures taken in the interviewed companies were, e.g., wiki-like pages for documentation of source code or services, as well as guides or check lists for processes and other important information related to performing work in a structured way. Another example was chat groups where individuals could ask, and answer, questions or talk across different teams or organizations, to facilitate the flow of information and better channels for sharing it. There is an obvious advantage to promote sharing within a team or organization as the collective knowledge of the processes and tools involved with the work being done, and the work itself, is more likely to accumulate over time. The dependency on individual people is lower, since the specialized or critical knowledge these people possess is being actively shared. In fact, the sharing could very well involve a large number of people and parties, e.g. beyond the DevOps teams themselves, there is management and other specialists within a company that could (or should) be involved (Rushgrove, 2016). Other characteristics benefit from sharing, e.g. collaboration and culture (Lwakatare et al., 2015; Smeds et al., 2015). In some cases sources might talk about "the culture of sharing" (Rushgrove, 2016), which indeed seems to be a way of working, and continuously learning and improving.

Feedback loops

To react to changes, and make changes more rapidly, quick feedback is important (section 2.3). The companies that were interviewed all had feedback loops, but in various degrees. In some companies, there could be processes with time-consuming asynchronous testing methods. These methods led in many cases to slow feedback. To compensate for the slow feedback and produce fast feedback again, the companies decided to use monitoring side by side. However, monitoring was not only used as a compensation of slow feedback, but was also used to increase and add feedback in the delivery lifecycle. Even if there were various degrees of feedback in the companies, all mentioned that feedback was an important characteristic, and that they in some extent strived to add as much feedback as possible (at least in every step of the process). Furthermore, feedback loops for an organization is essential to stay competitive and keep up with technological change, where delivering valuable and reliable products is important to keep trust and satisfaction among customers and stakeholders (Forsgren et al., 2019).

(Continuous) Integration

All interviews discussed continuous integration (chapter 2.3). There are differences in implementation depending on the type of company, e.g. companies who have tight restrictions (e.g. Smeds et al. 2015) based on security and stability, as well as producing accompanying hardware, had a harder time practicing this to the extent they wished, but in some places trials were being held to find ways of getting around this. Companies mainly focused on Software generally did not have as tight restrictions (but there are exceptions, of course, e.g.

financial or banking software where security and stability is important), and could practice this to a larger extent.

All interviewed companies regarded continuous integration as important in DevOps, and to integrate early and often. There can be many layers to what is called continuous integration, and other characteristics directly support it, such as automation (because a CI/CD pipeline is automated), and in turn CI supports other characteristics, such as QA and shifting left (because a CI/CD pipeline often contains automated testing, building, or other analysis tools being run).

(Continuous) Delivery

As with continuous integration, continuous delivery (chapter 2.3) is something most companies wanted to move towards, but differences based on the same reasons as in continuous integration restricted the implementation of it. Notable examples are cases where a full system composed of hardware and software needs to be validated according to strict rules and regulations, which creates a wall where one needs to wait for the other, and the following validation process is very long, which also affects implementation of changes, and further development on that specific project.

As mentioned in Continuous Integration, Continuous Delivery (CD) can bring positive effects as it streamlines the delivery process (Smeds et al., 2015). CD itself is the next step after CI, as you build and deliver a product or service. This process is to be automated, and through this automated process the goal is to keep the product or service in a deliverable state, meaning you are able to deliver at any point.

(Continuous) Deployment

Continuous deployment is the next step of continuous delivery (chapter 2.3) and deploys the product or service into production, and the restrictions that might exist on that step also restricts deployment as integration, delivery and deployment cannot be automated to the same extent in those cases (Smeds et al., 2015). Again, software companies who had few restrictions could build a pipeline automating this entire chain for parts of, or all of, their products or services.

(Continuous) Testing

As mentioned in all interviews, testing or rather continuous testing is needed in DevOps. The interviewed companies all strived after the same goals; to do it early in the process and as often as possible, with the help of automation. When products is expected to reach a certain level of quality, with few bugs, testing should be done early and under each step of the process. Also, it will be less expensive for a company to have continuous testing, than testing solely at the end of the process. In turn, continuous testing provides the ability to get fast feedback loops, continuous delivery, and continuous integration (Vega, 2019).

Planning

Mentioned in the interviews, all companies used some sort of planning in the form of project plans, release roadmaps, and delivery schedules. In the context of DevOps, working agile

there is sprint planning, which can be viewed as continuous planning throughout the development, as well as sprint reviews (Jabbari et al., 2016). In addition, information obtained from monitoring activities and other activities which provide information about development or the state of the product or service, such as testing, can loop back every iteration and be used in further sprint planning or decision making. However, the road for a successful DevOps adoption can be unique for each organization. The key is to learn from previous challenges and adapt to the future, which is done with the help of continuous planning (Smeds et al., 2015).

Change management

Mentioned as important in all interviews, and largely in accordance with chapter 2.3. Minor differences exist, mainly whether the company is more traditional (e.g. moving from waterfall towards a more agile approach) or more agile, where the strictness of the process itself can vary, but even in those cases there are exceptions. However, it's argued that having more formal heavyweight change process approvals have a negative impact on the delivery performance, where formal processes had no visible impact of lowering change failure rates. Instead organization should move away from these formal processes, and move more against (shift left) to peer reviewed-based approvals. These peer reviewed-based approvals in turn prevent, detect, and correct changes much earlier in the development lifecycle. They also had less impact on the performance, where letting employees work together improved stability and availability among teams (Forsgren et al., 2019).

Infrastructure as Code

Not as widely mentioned during interviews, but often in conjunction with cloud services, as in Infrastructure as a Service which is supported by Infrastructure as Code. The nature of Infrastructure as Code (chapter 2.3) is such that it is strongly supported by automation (Lwakatare et al., 2015; Smeds et al., 2015), which leads us to be confident in it being a good characteristic.

Some interviews also discussed Infrastructure as Code (IaC) being used for testing environments, where the environment could be version controlled and easily set up, and taken down through this practice throughout development. Beyond testing environments, most, if not all, actions performed on infrastructure can be automated, and this includes configuring and deploying or updating quickly, and eliminating repeated manual tasks (Lwakatare et al., 2015).

Agile and Lean

Agile (development) is a characteristic which was overlooked during the initial literature study, perhaps because it was so widely mentioned, which resulted in it being taken for granted by us. The agile manifesto lists the following principles (Beck et al., 2001):

- ***Individuals and Interactions*** *over processes and tools*
- ***Working Software*** *over comprehensive documentation*
- ***Customer Collaboration*** *over contract negotiation*

– ***Responding to Change*** *over following a plan*

These principles or practices are recurring in other characteristics listed, or supports them, e.g. culture, feedback loops, and collaboration.

Another new characteristic, which we have chosen to pair with agile development is Lean software development. Lean can be described through seven principles as follows (Poppendieck and Poppendieck, 2003):

– Eliminate waste

– Amplify learning

– Decide as late as possible

– Deliver as fast as possible

– Empower the team

– Build integrity in

– Optimize the whole

These principles are also recurring in other characteristics, or supports them, e.g. culture, sharing, automation, and collaboration.

As a whole, both Agile and Lean principles seems to heavily support the culture within DevOps, which leads us to believe this characteristic is important.

3.1 Ranking and visualizing characteristics and their dependencies

This section analyzes different ways of representing importance for each DevOps characteristic and their relations to each other, which will in turn provide us with valuable information for the following chapter.

3.1.1 Ranking of DevOps characteristics

To give us an idea of which characteristics were more important than others (if any), we tried to rank the DevOps characteristics based on our perceived importance (figure 3.1). However, when the ranking was completed, it felt like it did not really represent importance, but rather levels of abstractions. For instance, it was hard to compare Quality assurance with e.g. automation and monitoring, because of the differences in abstraction levels, where automation and monitoring are concrete tasks which are implemented, while quality assurance not necessarily is. Another thing that occurred to us was the question of why Quality assurance, planning, and infrastructure as code would be least important. Should this be interpreted that one can do DevOps without these? Probably not. All characteristics are important, and is needed to some extent when doing DevOps. Therefore, we felt that figure 3.1 gave

rise to more questions than answers, which led to us rethinking our idea, and find possible alternatives to represent the characteristics in relation to each other.

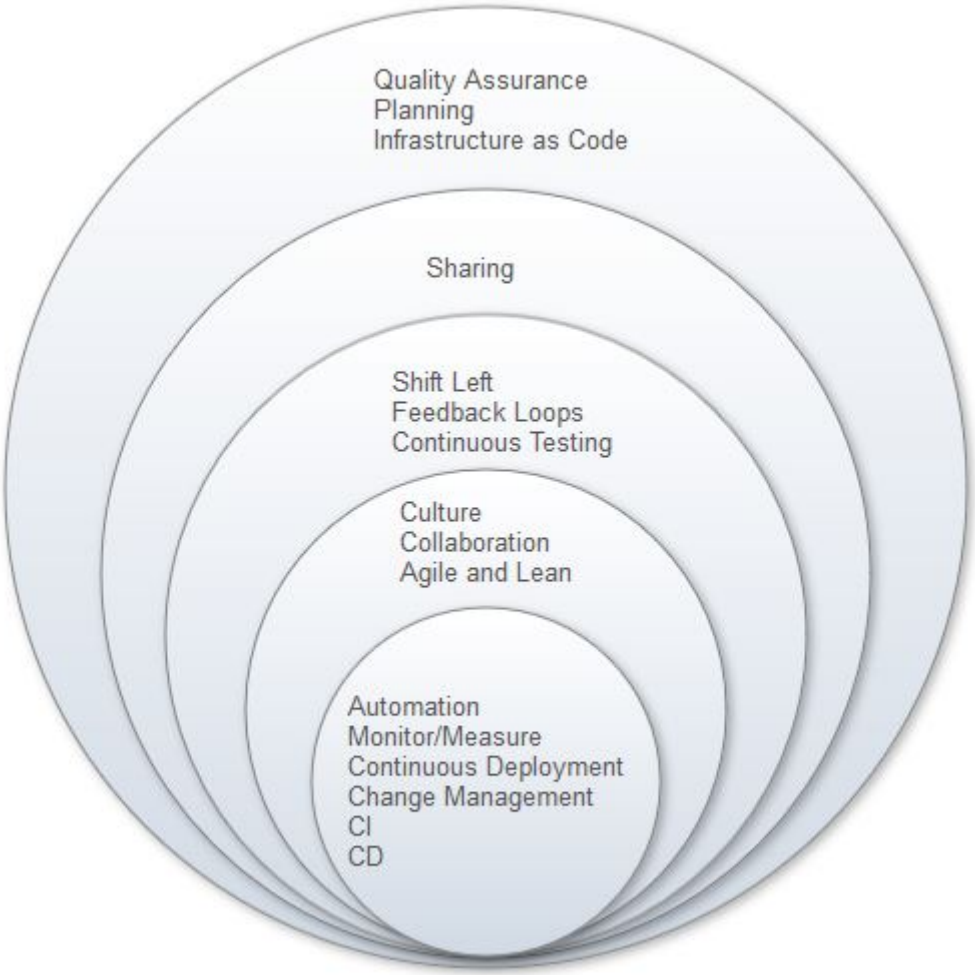

Figure 3.1: ranked characteristics of DevOps, where the smallest circle describes the most important characteristics, the second smallest the second most important characteristics, and so on

3.1.2 Dependency graph of DevOps characteristics

Because we did not agree with the ranking of the characteristics, we decided to explore alternate ways of representing importance. This resulted in a directed graph (Figure 3.2), where each vertex is a characteristic of DevOps, and each edge represents a dependency to another, as in the first supporting the other, or is required. An interesting observation in this graph is that some characteristics support many others, e.g. automation, whereas others are built by other supporting characteristics, e.g. culture and QA. These characteristics also differ in type, as automation is something you implement, and culture is more abstract, so the graph also shows different levels of abstraction of the characteristics, ranging from very concrete characteristics to more abstract ones.

There also seems to be different types of dependencies if the original definition is broken

down, such as a characteristic being a prerequisite for another, as in the case of e.g. automation and CI, CD, etc. as these characteristics contain automation as a way to perform them. If time was not constrained, these are some directions that would be very interesting to explore, but for the purposes of our thesis we feel that the graph is sufficient as is, and adequately shows our identified characteristics, and relationships between them, and rather than presenting a ranking of importance, provides our interpreted importance of each characteristic in relation to each other, as well as what one might need to implement a certain characteristic, or achieve a certain characteristic, and lastly that every characteristic indeed is part of the graph.

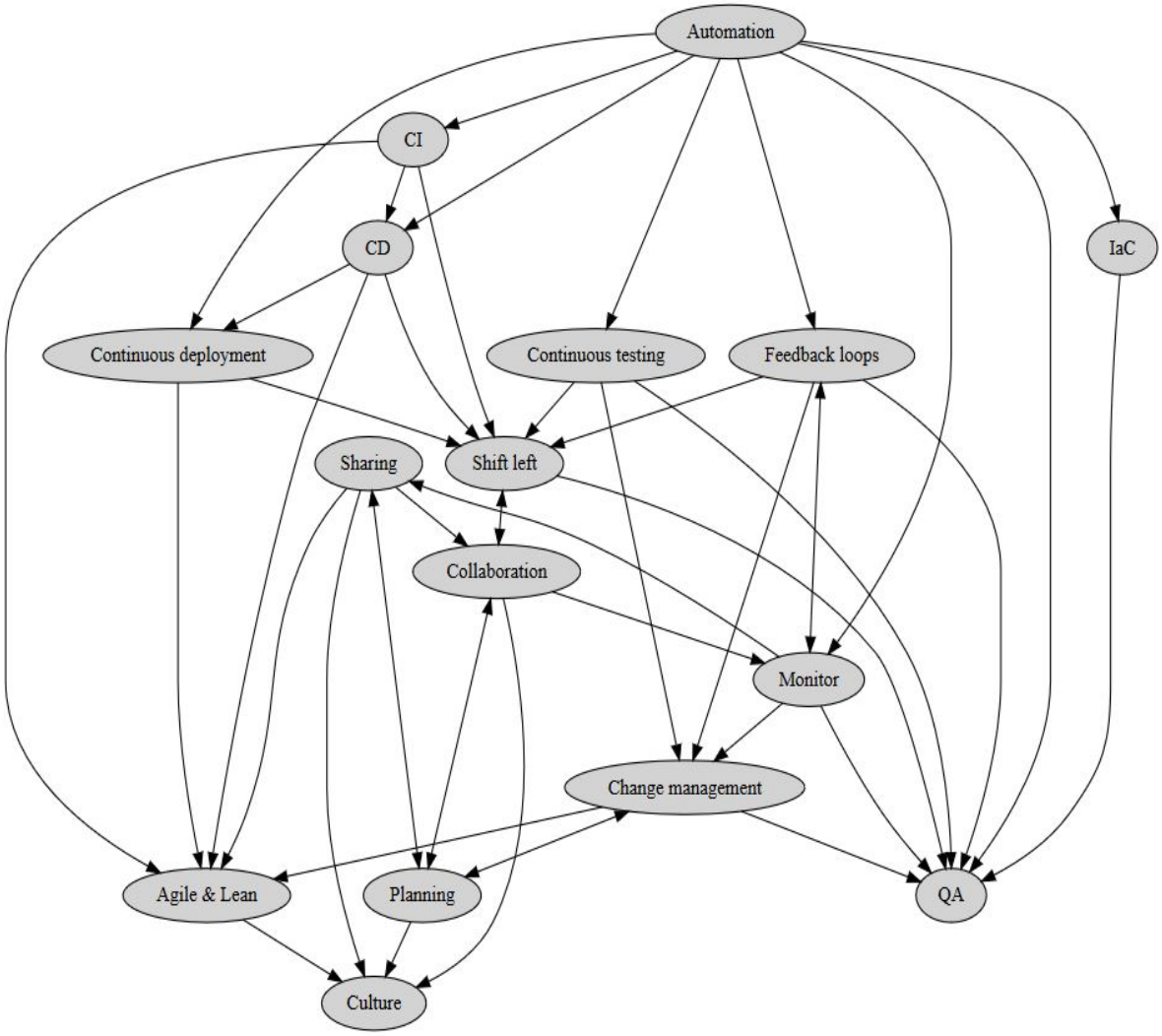

Figure 3.2: Characteristics of DevOps and dependencies between them

Example of how relationships were formed

Here we will provide a few examples of the reasoning behind the relationships between the characteristics in the graph above.

- **Automation** - As automation removes tasks from continuous human influence, and

formalizes tasks, its application is very wide. Examples of areas where one could automate steps or tasks are in a CI/CD pipeline, testing, feedback, IaC, etc. where tasks can be described in code, and therefore automated.

- **Continuous Testing** - Raises quality by continuously testing at different stages, and provides information and assurance regarding the state of the development. Being supported by automation as many tests can be automated, e.g. in a pipeline.
- **QA** - Is supported by characteristics which raises quality. Examples of such characteristics are testing and change management. Another example is IaC, which raises quality by describing infrastructure as code, which lets you version control, automate, and test it.

Chapter 4
SCM in a DevOps context

In this chapter we explore on how SCM could be used in a DevOps context, which SCM activities are needed and not needed in the DevOps context, and what new SCM concepts and principles can be added to the SCM toolbox. Through our interviews on SCM we will analyze how SCM could be used in practice, and investigate possible differences between the interviews responses and the initial literature study (chapter 2.4). This will in turn strengthen our classification of chapter 2.4. Furthermore, a graph will be drawn to clarify and visualize the connections and dependencies between SCM activities, which will help understand the internal sequences of the SCM activities. Moreover, to get an overview and visualize the mapping between the SCM activities, the DevOps characteristics, and the eight DevOps activities in the DevOps pipeline a graph will be drawn with these activities and characteristics combined. This will in turn help us produce the compiled result on how SCM could be used in a DevOps context, how it can be adapted in the DevOps context, which SCM activities is needed and not needed in the DevOps context, and what new SCM concepts and principles can be added to the SCM toolbox.

4.1 SCM in practice

Each SCM activity in this section is based on the responses from the SCM interviews. The section includes how the companies use different SCM activities, and possible differences between how the SCM activities are used in practice with how literature defines it (chapter 2.4).

Configuration Identification

All interviewed companies, except one, had some form of configuration identification activity. The company that didn't have a configuration identification activity already had an established product with established components, and didn't see a need for configuration iden-

tification. However, this product and these components was version controlled and had documentation, which implies that configuration identification had been done, and the product and components could be seen as configuration items. The other companies all continuously performed configuration identification, basing the new CIs on e.g. new requirements, redesign, and changes. Most of the companies also had an initial configuration identification, based on e.g. an initial idea of what is to be done, and/or by following requirements.

Configuration Control

As mentioned in all interviews, change control is seen as an important SCM activity, where both formal and informal change control processes took place. The processes could vary from company to company, but all change control processes was carried out with the help of CCB. However, the CCB decisions could be on different levels, where some decisions was taken on product owner level, and others on team level. In rare cases different experts (e.g. security expert) and configuration managers could take part of the CCB meeting. To decide the risk with the changes there had to be a risk analysis. This was done with the help of an e.g project managers/owners and development teams, where decisions were made about what impact the change had on the product (e.g other components and modules), and what value it added to the customer. The weight of the risk analysis could of course differ, where in some companies programmers or architects (team level) could decide if the change needed further impact investigation, and where in other companies a more bureaucratic processes took place.

Configuration Status Accounting

All companies had the ability to get information and status, e.g. metrics, about configuration items, and activities related to these, which led to full transparency among employees (e.g testers, QA and programmers). One could at any time during the lifecycle track information about a product. However, the traceability could vary from company to company, where some companies had so called epics, which was a big chunk of work that was broken down into features, user stories, and tasks. These, in turn, were linked together which created traceability. Other companies were somehow missing the traceability between documents, but was working on it. Nonetheless, all companies wanted to produce as much visibility for a system as possible, where, for instance, commits needed to be coupled with a tasks, changes sets needed to be trackable, and bugs needed to be coupled with features, user stories, tasks, and so on.

Configuration Audit

Not explicitly mentioned during the interviews, but the general meaning was present in most, as in chapter 2.4. All companies did testing before baselining a product. These tests were carried out to see that all modules/components in a product were working in relation to each other, which could both be hardware and software. In most companies, documentation based checks were also carried out to see that all features and stories had been met. To make this process as smooth as possible, test case links to features and stories were created, which in turn provided important traceability for tests. The audits could both be physical and

functional. However, something that was not mentioned in any of the interviewed companies was if there were a physical person who decided if the product should be released or not. In most cases, if bugs or errors were detected in an audit they were either corrected in a roll forward approach or disabled from the release by removing the corresponding feature.

Version Control/Collaboration tools

All interviewed companies used version control and collaboration tools. They all had some kind of distributed version control tool, which automatically tracked changes in the source code (e.g. git). Therefore, In the software part, most things if not all was version controlled. Also, most of the documentation was version controlled. However, there were some companies that had problems of version controlling the team documentation (the internal documentation), where the extent of the version control could vary from team to team. One company was lacking version control on user data, monitoring, and testing. However, they wanted to improve this and strived, just as the other companies, to accomplish as much version control as possible.

Build Management

All interviewed companies had some way of specifying a system to be built, some mentioning build optimizations for production. Many used different types of build management tools to build, and specified the system in a bill-of-materials like document, especially in cases where software and hardware were closely tied. In some cases this document also contained information on how the system was to be built, and some companies mentioned the usage of make-files. All companies could handle variants of systems through their process, but some expressed that variants were not common for their products, and so the need was not there.

Teamwork/Parallel work

Most companies described agile ways of working. In addition they all used git for most of their version control, which also helps enabling collaboration and parallel work. They all described teams as having collective ownership and responsibility. Other aspects of teamwork mentioned was e.g. communication channels for asking and answering questions and sharing information between teams. For enabling parallel work, all companies had adopted certain branching strategies when working within a shared repository. The branching strategies used were e.g. trunk-based development, integrating early and often, and variations of git flow. Some companies only expected teams to deliver to a master branch, while leaving the specifics of exactly how development was done to the teams, while others described differences in team maturity, where a mature team might work with trunk-based development, and a less mature team might adopt a more git flow like development strategy. In the latter case, the company did supply recommendations on different strategies discussing pros and cons of each, and ultimately letting the teams decide.

Workspace Management

All the interviewed companies used version control tools with functionality to manage individual workspaces, e.g. git which also (under normal usage) enforces practices to counter

problems listed in chapter 2.4, such as the shared data problem and simultaneous update problem, for example by making sure the personal workspace is up to date with the repository before anything new is added. Many mentioned using pull requests when adding changes to a common repository, instead of e.g. pushing a change in git, and often in conjunction with a code review process, which differed somewhat between companies. As an example, one large company producing both hardware and software needed the approval of a configuration manager when adding changes to the master branch, and internal (within the team) approval when adding to production, while other companies left it to the teams to decide how they wanted this to be done.

Change Management

Change management was not explicitly mentioned during the interviews, but the general meaning was covered (section 2.4), where all the interviewed companies tracked and had traceability between the origin of a change and the implemented source code. Generally, the change management process included a code commit, which was coupled with a task, and where the task was coupled to user stories and features. Tools used for change management were, among others, BitBucket, Jira, and Confluence.

Release Management

All interviewed companies had some form of auditing (see chapter 2.4) throughout the development process, which led to a baseline of the system. They also documented CIs went into a build, e.g. in a bill-of-materials like document, and also how it was built, which enables the recreation of previous releases. Companies producing both hardware and software generally documented this information with a higher granularity, or wished to do so, some citing e.g. external CIs (dependencies, e.g. external libraries) as a potential vulnerability, and the need to properly keeping track of them, or software closer to hardware being more sensitive to changes.

SCM Plan

Few companies had an SCM plan describing their SCM activities, and how they should be performed. One company in particular extensively employed the usage of SCM plans, with the motivation that it is very important in order to maintain a product. Another company planned and documented activities related to certain other activities, e.g. testing, for traceability, and strategies for saving data. One company had processes which stated it should be done, but currently lacked the tools and knowledge. Lastly, two companies did not employ the usage of SCM plans at all, one stating that within the company there were established processes, and generally the teams knew what to do, but added that for some activities there were checklists, and the other that it would be good to have.

4.2 Dependency graph of SCM activities

Because the mapping of characteristics in the graph of DevOps characteristics in section 3.1.2 produced a very clear view of how the characteristics could be connected, we employed

the same technique to clarify the connections or dependencies between the different SCM activities, in order to be consistent, and produced a graph showing these connections based on our own analysis (Figure 4.1). In the same way as in chapter 3.1.2 the edges shows an activity supporting another, or is required for another. An example of this is change management which requires some form of configuration control, which in turn requires the ability to retrieve information about the system and its CIs through some form of configuration status accounting.

An interesting observation in the graph is that three subtrees emerge from the connections. One can view these as part of an iteration of a development cycle where you begin by planning your SCM activities, followed by changes being made to an emerging system (left subtree), followed by building and releasing (middle subtree). The left subtree is most likely a process that loops throughout development, but we decided not to add in looping edges to avoid clutter in the graph, but be aware that these exist. The right subtree contains activities that enable and support other activities and development, and these activities also happen continuously in parallel with the other activities in the graph.

Much like for the DevOps graph in chapter 3.1.2, this graph is not an exhaustive mapping of connections and dependencies between the SCM activities, as many types of connections and dependencies most likely exist, not limited to our definition above, and even with our definition this is also the case. Because of time constraints we decided it was not feasible to try to produce an exhaustive mapping of these activities, and instead focus on showing that there indeed are connections between them, and that the activities produce a connected graph, where no single activity is unconnected.

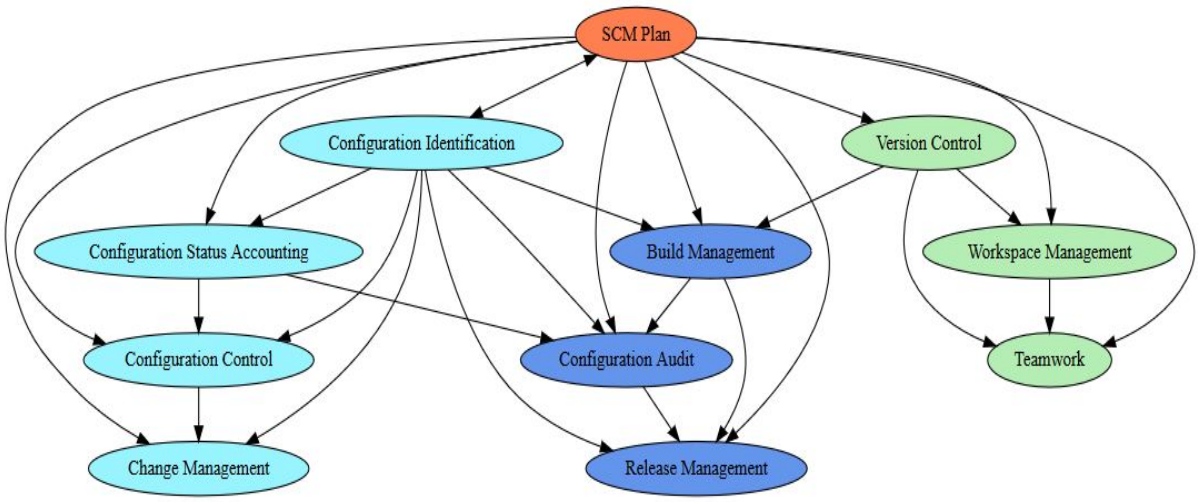

Figure 4.1: Connections and dependencies between SCM activities

4.3 SCM and DevOps combined

With the two previous graphs (graph of DevOps characteristics in chapter 3 and SCM activities), the internal connections of SCM and DevOps have been clarified, in isolation, within their own contexts. To further build a model that can help us explore SCM in DevOps, and ultimately answering our research questions by relating an SCM activity to a subset of DevOps

characteristics (and by extension also their internal connections), we decided to use the same approach for identifying connections between SCM activities, individual DevOps characteristics, and DevOps activities, in much the same way as in the previous graphs, which resulted in the graph in this section (Figure 4.2). From the DevOps characteristics, edges have been connected to activities of the "DevOps-eight", which is a common way of representing the DevOps cycle, with some variations. This was done so that the DevOps characteristics can be associated to a broader activity, and also because of the widespread use of this representation. In order to make the graph as clear as possible, the internal edges within SCM and DevOps have been omitted, but keep in mind that these still exist, and this graph only shows the edges between SCM activities and DevOps characteristics. The edges from an SCM activity represent a subset of an SCM activity being present in a DevOps characteristic, or that it likely is used there, or could be, in some capacity, based on e.g. its definition and interviews, and because we focus on SCM, the edges have been evaluated from an SCM perspective. Just like for the two other graphs, an edge can represent many types of relationships, but within the scope of this thesis (and again because of time constraints), the definition of these edges have been chosen to represent the relationships described above. The specifics of the mappings present in the graph will be discussed in more detail in the next section.

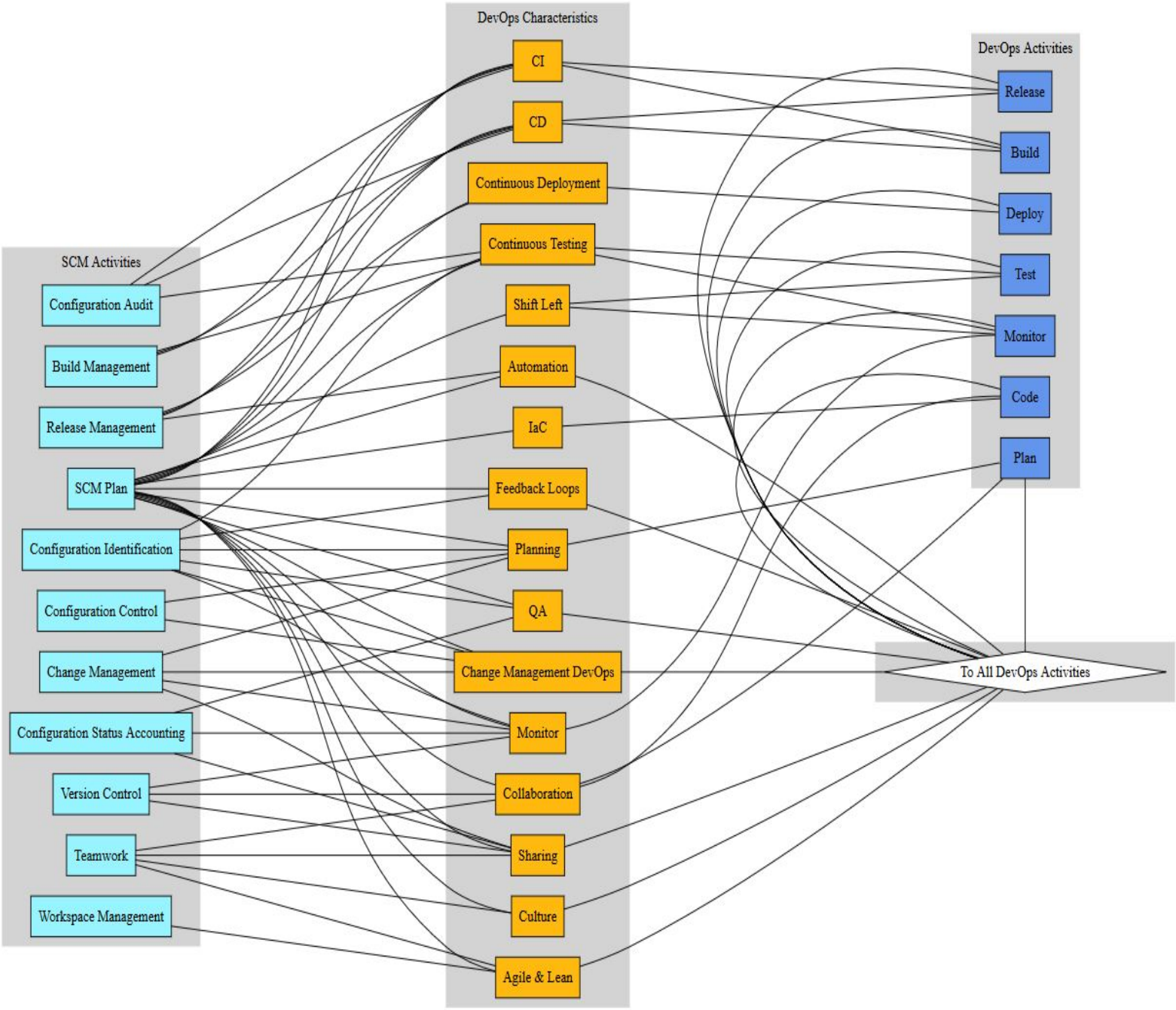

Figure 4.2: Connections between SCM acitvties, DevOps characteristics and DevOps Activties

Example of relationships were formed

Here we present a few examples of how SCM activities were mapped to DevOps characteristics in the graph above.

- **Configuration Audit** - In a *CI/CD* pipeline *(continuous) testing* is performed, which functions as part of an audit, as the tests mirror expected behavior, requirements, etc.

- **Configuration Control** - Configuration control describes the process by which changes are to be handled. As such, *planning* is involved, both in deciding on the process, and in its execution through *change management*.
- **Build Management** - Builds are performed in a pipeline (e.g. *CI* and *CD*), which also contains *testing* the build.

4.4 Analysis

This section analyzes the various connections in figure 3.2, 4.1, and 4.2. In this case, each SCM activity is seen as the starting point, but note that the starting point could also be the DevOps activities or the DevOps characteristics. Along with the SCM activities each graph is analyzed, where both internal and external connections are described. This in turn increases the understanding for the graphs, and provides valuable information about why or why not the SCM activities are needed in a DevOps context.

Configuration control

Following the graph in section 4.3 (Figure 4.2) from the SCM activity configuration control, it connects to the DevOps characteristics change management and planning, which in turn connects to all of the DevOps activities, with an overlap on plan.

The DevOps characteristics in this sequence have edges in both directions in the DevOps graph, with the reasoning that change management requires planning, and the usage of change management supports planning throughout development. These two characteristics both have an impact on the activities of DevOps, directly or indirectly, because change management is continuously used in each step, as feedback is present in each step, which can prompt a change anywhere. Planning also affects the whole cycle, as it is both a continuous process, as well as more "formal" activities, e.g. a sprint planning.

Configuration control describes a process for handling a change, and this process can vary, as seen in the interviews ranging from formal CCB meetings and thorough impact analysis, to code reviews or discussions within a team, which can also be viewed as a form of CCB meeting and analysis for deciding on a change, its impact, implementation and validation. This process is a supporting activity in the SCM graph to change management, as it is the process that produces changes to be managed. As such, it relates to the previous DevOps characteristics through a common overarching activity, change management, which itself requires planning, and in turn produces information that can be used in planning.

Change management

From the graph in section 4.3 (Figure 4.2) the SCM activity Change Management connects to the DevOps characteristics change management, planning, monitor and sharing. These characteristics in turn connects to all of the DevOps activities with an overlap on Plan and Monitor. A reason for these connected characteristics is that change management requires planning in order to cover all changes to a system, as discussed in chapter 2.4. In DevOps, monitoring activities should be employed early in the process if shift left is followed, and the contents of what is being monitored is up to each team. Several of the interviewed companies

who used monitoring also tracked and monitored changes, with traceability built in, and in some cases visualization of the data. Because this monitoring data is (or should be) available to the whole team, and often parsed to a more human friendly state, the DevOps characteristic of sharing can also be said to be employed. Although change management in DevOps was quite vaguely described in chapter 2.3, as the total description needed to be collected from many sources, there seems to be a large overlap between DevOps change management, and SCM change management regarding what it wants to accomplish, as could probably be expected. We have identified characteristics within DevOps which happens to be of great help for collecting, tracking, managing, visualizing and spreading this information, and these characteristics or processes together form a basis for change management within DevOps.

Configuration identification

From the graph in section 4.3 (Figure 4.2) configuration identification connects to the DevOps characteristics change management, planning, monitor, QA, feedback loops and continuous testing, which in turn connects to all the DevOps activities with an overlap on Plan, Monitor and Test. Since configuration identification is a process for identifying configuration items of a system, and their features, it is no surprise that change management in DevOps is affected, as change management handles changes on configuration items which are identified using a configuration identification process. Through continuous planning, aspects of a system can be made visible, and new features or components can be identified, which is the more continuous aspect of configuration identification, as described in interviews. The configuration items one has affects what is being monitored, as the monitoring activity does not have to be exclusively for deployed products. Both monitoring and testing produces results which can, through feedback, potentially be used to identify needs, both in the form of changes and new components, and also removals, throughout development. Through these characteristics there are many aspects that provide a basis for configuration identification throughout the development cycle, and also leading up to it, through planning where configuration identification might be present. Since DevOps wants to be open to change, the continuous and iterative aspects of the affected characteristics would also support a continuous and iterative configuration identification process, much like what has been discussed in interviews.

Configuration status accounting

When following the graph in section 4.3 (Figure 4.2), we can see that the SCM activity Configuration status accounting connects to the DevOps characteristics Monitor, QA, and Sharing, which in turn connects to all of the DevOps activities, with a focus on Monitor.

In the DevOps graph there is an edge from monitor to QA, with the reasoning that QA is supported by monitor. This because, when having monitor integrated early in the process there is a high chance of finding service failures without delays, which in turn provides better QA. Monitor also includes making data available for both Dev and Ops, and thus support sharing (edge from monitor to sharing). All these three characteristics have an impact on the activities of DevOps. Both sharing and QA are characteristics that are used in each step of the seven activities. The characteristic monitor has an edge to the DevOps activity monitor, where the monitor activity is something that is used continuously during the DevOps process,

and therefore affects the whole cycle.

Configuration status accounting is described as the ability to get metrics about configuration items, and activities related to these. In the SCM-graph, Configuration status accounting supports configuration control and configuration Audit. This because of configuration control and configuration audits need of getting metrics about configuration items. In the interviews, all companies used some sort of configuration status accounting to get full transparency among employees, and provide the ability to get metrics at any time during the life cycle. This in turn can be coupled with the DevOps characteristics sharing, QA, and monitor, where all have the need of getting information about metrics, and where monitor and sharing also provides the ability to share metrics.

Version control/Collaboration tools

The node version control in figure 4.2 connects to the DevOps characteristics monitor, sharing, IaC, and collaboration, which in turn connects to all DevOps activities, with an overlap on monitor, plan, and code.

We can see that in the DevOps graph (Figure 3.1) the characteristics monitor, collaboration, and sharing have a circular dependency, where monitor support sharing, sharing support collaboration, and collaboration support monitor. As mentioned in the subsection configuration status accounting, monitoring supports sharing because of the ability for Dev and Ops to take part of data through, for instance, dashboards. In turn, sharing support collaboration due to that sharing information between employees, which leads to more awareness, which in turn creates better collaboration. When extending the collaboration between development and operation personnel, there will be a higher degree of sharing, which means that collaboration also supports sharing (Figure 4.2). Furthermore, to integrate monitoring early in the DevOps process there needs to be some sort of collaboration between dev and ops, thus collaboration supports monitoring. All these DevOps characteristics have an impact on the DevOps activities (Figure 4.2). For instance, there needs to be a plan that describes which version control tools to be used in the DevOps process. The plan is something that is done continuously during the DevOps process. Also, version control helps the ability to do rollbacks (although the general mentality in the interviews was to “roll forward”), and prevent concurrent work from conflicting, which can be included in, for instance, the DevOps activity code. Moreover, all companies that were interviewed strived to version control as much as possible, which probably means that version control should be included in every DevOps activity. Also, version control is a valuable asset that manages and keeps track of the history of configuration items. Configuration items are created and managed throughout the devops process (including infrastructure, as is evident by the edge to IaC, e.g. configurations for server environments, test environments and other parts of one’s pipeline, e.g. jenkins configurations), and should therefore be version controlled during each step of the DevOps activities.

Configuration audit

The SCM activity configuration audit connects to the DevOps characteristics continuous testing and CD, which in turn connects to DevOps activities test, release, monitor, build (Figure 4.2). The reasoning behind the connection between configuration audit and CD is

simple, to ensure that the product is in delivered condition, there is a need to see that the product works in accordance with changes, specifications and requirements. Although configuration audit was not explicitly mentioned during the interviews, the general meaning was present, with all companies having some form of configuration audit on products or services that had reached a sufficient level. Since, CD includes building and deliver a product or service it will in turn connect to the DevOps activities build and release. Furthermore, Configuration audit can be seen as a subset of continuous testing, in which configuration audit tests are performed in conjunction with release and baselining, and where continuous testing in the CI/CD pipeline contribute to release and baselining. Also, by going from continuous testing to configuration audit, we can see continuous testing as a support for configuration audit, where testing must be performed regularly to ensure the deliverable condition of the product or service. Continuous testing then connected to the DevOps activities test and monitor, where testing and monitoring must be done throughout the DevOps process to detect, for instance, bugs and errors.

Build management

From figure 4.2, build management connects to the characteristics continuous testing, CI, and CD, and then to Test, Monitor, Build, and Release in the DevOps activities. The reasoning for the connections to the characteristics is that in DevOps the building is done in the CI/CD pipeline, and within that pipeline the build is tested, and this naturally extends to the listed DevOps activities for the same reason. Since build management is not only the process that handles building a system, but also defines the system to build, including all components and dependencies, there needs to be such an aspect present. Throughout the interviews it was common to specify a system in some way, most used tools where each build was recorded in varying degrees of specificity, e.g. a tool mentioned for this was docker. Since the building is in many cases automated, or should be if a CI/CD pipeline is used, it can be argued that build management in essence is very compatible with these DevOps characteristics, as the automated build process still needs to know what to build, and that information needs to be recorded somewhere, either through the build tool itself, or before (and act as input to the build process). Some companies mainly recorded built systems or releases, and maintained that it would be possible to recreate these releases (with varying degrees of difficulty, e.g. regarding how good traceability there is present) although the exact input was not necessarily recorded, while other companies maintained very specific BoM-like specifications for each version/build making this process very simple.

SCM Plan

In figure 4.2, SCM Plan is connected to all DevOps characteristics, and is because of this also connected to all DevOps activities. The reasoning behind this is simple, as there are potentially SCM activities present in all DevOps activities, and an SCM plan describes what activities are used where, and how they are to be used. This is also further strengthened by the fact that through the rest of the SCM activities, there is practically full coverage of DevOps characteristics, and all DevOps activities. Although far from everyone in interviews explicitly used an SCM plan, there were often documents or other information sources available which provided information on how certain things should, or could, be done, including SCM

activities or tasks related to an SCM activity, showing that although it might not be called an SCM plan or contain only SCM related material, or even be a single document, something similar is often used.

Release management

In figure 4.2, Release management connects to the DevOps characteristics CI, CD, continuous deployment and automation, which in turn connects to all the DevOps activities, with an overlap on build, release, and deploy. Release management handle both formal and informal releases to customers, and the reasoning behind release management connections to CI, CD and continuous deployment is that all these steps are included (handling everything from code integration to deployment) when products or services are released to customers. The reason behind the automation is that this chain of parts should be built as an automated pipeline, at least for companies with few restrictions. In addition, automation should take place where possible in the DevOps cycle to produce reliable and repeatable processes, which in turn includes all DevOps activities. Although release management affect the whole process, the focus (overlap) is on the DevOps activities build, release, and deploy, which are the three critical steps for a release to take place.

Teamwork/Parallel work

The SCM activity Teamwork connects to the DevOps characteristics sharing, collaboration, culture, and agile & Lean. These DevOps characteristics in turn connects to all DevOps activities, with an overlap on Plan and Code. The reasoning for connecting teamwork to these characteristics is because they all touch on things that involves teamwork, and their activities as well as principles and practices (e.g. in agile & lean). The overlap on Plan and Code is likely because planning in an agile context such as DevOps is a team effort, and the same is true for writing code as a team, as you need to collaborate, and be able to collaborate and be able to work in parallel through a common set of tools and coordinate tasks, decide on branching strategies, communication channels etc. The reason all of the DevOps activities are covered is because teamwork and parallel work can affect every step of the cycle, because the team as a whole is responsible for their work, and that includes everything that enables their teamwork, collaboration, and parallel work.

Workspace management

In figure 4.2, workspace management connects to the DevOps characteristic Agile & Lean, which in turn connects to all the DevOps activities. Workspace management is a key component of Agile & Lean, where it both creates flexibility for developers to experiment freely (locally), and creates the ability to publish code when ready for it. Also, workspace management support a range of different workflows and, as mentioned in the interviews, these workflows is often up to the teams to decide, and once the workflow has been merged with the master branch the agile workflow is done, which contribute to efficiency. Furthermore, workspace management creates the ability for changes to be pushed down the deployment pipeline faster than working with monolithic releases and centralized version control systems, which helps agile teams to move faster. All interviewed companies had code reviews,

which could be managed by individual team members or configuration managers. These code reviews, in turn, increases confidence that the code being published works as it should and is ready to be released, which is likely to increase confidence to releasing more often, and therefore being more agile. As mentioned, Agile & Lean then connects to all DevOps activities. The reason behind this is Agile & Leans strong support in culture, where culture in turn has important aspects of DevOps as a whole.

4.5 Result

This section provides a detailed description of the results in relation to each research question using previous analysis and data collections.

4.5.1 RQ1a: What SCM concepts and principles are needed and not needed in a DevOps context?

This section goes through each SCM activity and determines whether it is needed or not in the DevOps context. The results are based on literature, interviews and own analysis, where table 4.1 gives an overview of what SCM activities are needed and not needed.

SCM activity	Audit	Build	Release	CM Plan	CC	CM	CSA	VC	Teamwork	Workspace	CI
Needed	x	x	x		x	x	x	x	x	x	x
Not needed				x							

Table 4.1: Shows which SCM concepts are needed and not needed in the DevOps context

Configuration audit

All interviewed companies used some form of audit. The audit could be in line with the definition in section 2.4, where it was carried out when a product or service had reached a sufficient level to ensure quality (Bendix and Ekman, 2007). However, as not explicitly mentioned during the DevOps interviews, but the general meaning was present, audit could also be carried out in a DevOps fashion, where it was used in combination with a CI/CD pipeline. Each test in the CI/CD pipeline could be seen as an evaluation of the product or service, which in turn reflects the configuration documentation and change request that must be fulfilled for a release or baseline. Although, this DevOps approach may differ from the configuration audit definition in terms of time and automation, all indicates that configuration audit is a needed SCM activity in the DevOps context.

Build management

Since build management is the process which defines a system model and how to build it (Bendix and Ekman, 2007), it was clear throughout the SCM focused interviews that each company met these requirements by the build management tools they used. In the DevOps focused interviews, all had an automated pipeline which included continuous integration and

continuous delivery steps, and if these are present, the system is also built in the pipeline, which implies that a system model, and instructions for its build, is also present on some level. Therefore, build management is evaluated to be both present in DevOps and needed, especially because the build stage of the pipeline is automated.

Release management

Release management needs physical and functional audits before a release to verify the correct system has been built. Also, a bill of material should be used to record how the release was built, and what it consist of (Bendix and Ekman, 2007). In the SCM interviews all companies except one followed these requirements for release management. The company that didn't follow the definition in section 2.4 did a rebuild every time the application was deployed, which was not version controlled. According to them, this was done for optimization reasons, and nothing had gone wrong, which made it hard to argue against. However, there was an overlap between DevOps and the companies that followed the requirements for release management. In the DevOps interviews, all except one company version controlled the bill of materials and had audits before a release. In the CI/CD pipeline each check-in produces a releasable state, and so the state of a repository after each check-in could be seen as a bill of material that was version controlled, and where the test in the CI/CD pipeline could be seen as an audit. Therefore, we believe that release management is needed in the DevOps context.

SCM Plan

An SCM plan describes SCM activities, and is used as a reference for the planned SCM activities, spreading awareness and information about SCM, and the planned activities in particular (Leon, 2014). Although we do think this has high value in DevOps, where its existence and goals even overlap with DevOps characteristics, e.g. sharing, and its presence in the SCM focused interviews, we feel the specificity of an SCM plan might be too high in a DevOps context, and could possibly be at risk of being regarded as a disconnected component of the development cycle. An alternative to this is discussed in section 4.5.2, where a DevOps process plan is introduced as an alternative, where the contents of an SCM plan is a subset of this overarching plan.

Configuration identification

Since, configuration identification refers to identifying distinct components of a given system (Bendix and Ekman, 2007), it felt as a necessity that all companies used configuration identification, which they did (both DevOps and SCM interviews). Without configuration identification there would most likely be no identified components of a system, which would lead to difficulties in knowing which components to use, when to use them, and how to use them, which in turn would create questions such as, how do we build, what does the product consists of, and so on.

Configuration control

Even if the configuration control process could vary from company to company, all interviewed companies with the focus on SCM met the requirements for it, where they both had

informal and formal change processes, which was carried out with the help of a CCB. Since, the CCB control performs the evaluation of a change, where an impact analysis take place to understand the impact the change has (Bendix and Ekman, 2007; C&A, nd), it felt as an important aspect in DevOps. In the DevOps interviews, we could see an overlap between the DevOps change process and configuration control. As in the SCM interviews the change process in DevOps could vary, but there was a CCB, which might be argued to be more agile, and in many cases on team level. However, the general meaning was present, and one can argue that without a configuration control, there will be no impact analysis, no traceability, no validation of implementation/testing, which is not very good, and can lead to unpleasant surprises.

Change management

If change management should be able to track changes from origin to implementation through tools and processes (Bendix and Ekman, 2007), other SCM activities needs to be present to provide this information, such as configuration control, which formalizes a change process (Bendix and Ekman, 2007; C&A, nd) and tools to manage tasks, collaboration, and version control. Throughout most interviews it was clear that most companies had tools and processes overlapping with these activities, and expressed a need for, e.g., the traceability it provides. As such, it is both present as a consequence of the combination of tools and processes being used, as well as needed through the information it provides and maintains.

Configuration status accounting

The ability to get metrics regarding configuration items, and activities related to them (Bendix and Ekman, 2007; Daniels, 1985), is described as possible for all interviewed companies. Some companies directly tracked such information through their monitoring activities, which helps create visibility to an even higher degree, as parts of the status accounting can be said to have been automated. Without some form of status accounting (and in extension other supporting activities, such as change management, configuration control, version control, etc.), it would probably range between impossible and tedious to extract and organize project information as described above, and since the output from status accounting can be used in other activities such as auditing, planning, etc. we have rated the need to be high.

Version control

Version control was mentioned as a valuable asset by all the interviewed companies, where version control tools provided support for storage, versioning, and traceability for configuration items (Bendix and Ekman, 2007; Asklund et al., 2004). Therefore, the need for version control could be seen throughout the DevOps process, where many activities depended on it. For instance, version control provides automatic support for configuration status accounting (Bendix and Ekman, 2007; Asklund et al., 2004), which need was rated high in the DevOps context.

Teamwork/parallell work

As teamwork and parallel work touches on interaction/communication, coordination, collaboration, both as a team and for working in parallel, it puts emphasis on planning these aspects (Appleton et al., 1998), and agree as a team. In the interviews all companies described processes and tools that supported and helped activities like these, in addition to describing themselves as varying degrees of agile, as well as methods of communication within teams and between teams, and internally had guidelines or suggestions for different branching strategies, with the teams often having a high degree of freedom and responsibility. Ultimately, teamwork and parallel work needs to be planned out to be able to provide a stable basis and flow throughout a project, and is therefore needed.

Workspace management

Workspace management could be seen as a side effect of the version control tools, where version control tools (e.g git) provide the ability to make a copy of the chosen documents or modules, make the changes, and then add the change to the repository. It should not be allowed to work directly from the repository, because of Wayne Babich shared data problem. However, this can be avoided with Workspace management, where workspace management let developers work within a controlled environment (Bendix and Ekman, 2007), which is a necessity in the DevOps context.

4.5.2 RQ1b: What new SCM concepts and principles can be added to the SCM toolbox?

As mentioned in section 4.5.1, we believe the scope of a "traditional" SCM plan (Leon, 2014) might be too narrow and specific for DevOps, and might separate SCM activities from DevOps activities instead of seamlessly integrating them and letting them be part of the work flow.

Most companies we have interviewed had resources or documents describing parts of their development process, or practices, how to do various things as a team, etc. And this is collectively a plan for, e.g., teamwork and collaboration, version control, change management, and many other aspects of both DevOps and SCM.

We propose a combination of what we have encountered throughout our interviews with focus on both DevOps and SCM, by collecting these resources and/or documents and check lists, into a "Process Plan". This plan could initially contain a description of each step in one's development cycle, how these activities are to be performed, what tools to use and how to use them, and also the SCM aspects of them, and additional SCM activities outside one's development activities if one would choose to separate activities for some reason. Essentially the plan is an SCM plan, where the SCM activities are put into relation to the specific development process a given team or organization chooses to work with, and gives information about who, what, when, and possibly the most important, why, a given activity is performed. Because the overall development process will also be described in this plan, it will function as a guide or instruction for the team or organization to follow. Because this is in the context of DevOps, the plan itself is treated as a living document, which is open to change, and its existence supports the DevOps characteristic sharing, and empowers the team and promotes

responsibility and continuous learning. Also, because it will be a collectively used document, the information within will be a part of the DevOps culture, and helps promoting it throughout the team or organization.

4.5.3 RQ1c: How to adapt relevant SCM concepts and principles in a DevOps context?

In this section guidelines will be presented based on the analysis in section 4.4. The guidelines will discuss how each SCM activity can be adapted and used in a DevOps context, in a structured way. In section 4.4 it was shown that each SCM activity covered more than one DevOps activity, which implies the SCM activity is continuous throughout several Devops activities, or parallel to others, which makes sense since many DevOps activities are parallel and continuous in nature. This will be reflected in this section, but when discussed only the activities we have determined to be most critical will be mentioned. The guidelines will be based on our previous literature study, the interviews, and the analysis in chapter 3 and the sections earlier in this chapter. The guidelines will also feature examples of how each SCM activity can be achieved in a DevOps context with the same basis as the guidelines themselves.

Configuration control

Planning is an important part for configuration control. The goal of planning the configuration control is to develop a process to be used. In the context of DevOps the focus should be to keep the process agile and flexible, and this means e.g. that one would favor informal meetings rather than formal. When planning one should also agree on, and specify, who is involved in the steps of the process, and when. For example, to focus on a lightweight and agile process, the team should be responsible, which means the team itself could act as an informal CCB, where the collective knowledge of the team is the basis for impact analysis, decisions, and validation of changes. To increase the stability of the process, there should be as little reliance on any singular person as possible, to reduce internal handovers and leaving decision making to a specific person. Though these more agile adaptations, the steps of configuration control, as described in e.g. (Bendix and Ekman, 2007; C&A, nd), are still maintained.

Change management

It was evident that several DevOps activities together laid a foundation for change management. A large part of that is because of monitoring, which in interviews was used to track (amongst other things) changes, and show their connections to a certain change request, and CI, with the help of tools supporting this. An example of this within the context of DevOps could be to track issues and tasks in relation to a CI or a collection of CIs, where a relation is always enforced, and any change request is always in relation to a CI, or one/some of its issues or tasks, or requirement. In interviews we have seen this in varying degrees, and a common factor for failure was any step that needed to be handled manually or if the issue tracker did not strictly require a relation for it to be added. From this, the needed functionality is to be able to preserve traceability from a change request, and its associated task/issue/CI, to its implementation (Bendix and Ekman, 2007), e.g. by connecting a check-in/push to an existing task or issue, and through monitoring making this information available to the whole team.

Configuration identification

Configuration identification has a focus on the DevOps activities plan, monitor, and test. The purpose with planning the configuration identification is to identify distinct components of a given system (Bendix and Ekman, 2007). In the DevOps manner, this plan should be agile and flexible, where one should have a continuous planning process for identified CI:s during and at the beginning of the DevOps process, which can be done with the help of early and continuous monitoring and testing. However, it is always a risk to over do the CIs, which adds complexity for no value. This means that there is a need for limiting CIs, where the granularity of a CI should be determined by the value it adds (Kelly, 1996).

Configuration status accounting

Configuration status accounting is an important part of the DevOps activity monitor. Configuration status accounting provides the ability to collect metrics about configuration items (Bendix and Ekman, 2007; Daniels, 1985) that can then be used for monitoring. Monitor in turn contributes to increased transparency and visibility among employees, which were, according to the interviewed companies, two important aspects of configuration status management. Configuration status accounting should be implemented with the help of CMDB, so that one could at any time during the life cycle get and track information about a product. Monitor can then be incorporated early in the process, and be provided with valuable metrics, which in turn increase the transparency and visibility among employees. Also, by doing this, the DevOps process can be continuously provided with valuable feedback.

Version control/Collaboration tools

The SCM activity version control(/collaboration tools) touches upon all DevOps activities, as all code (and documentation and other related objects) are potential candidates for being version controlled. Planning version control would likely be a positive initial step (but also something that can change over time, if new requirements or needs arise within a team), where a team decides which tools are to be used for this purpose, and how they should function together. Part of this planning activity could be to focus on understanding how to use the tools in order to achieve the goals of the team (and making sure that they can), and how to avoid issues that might arise when collaborating on a shared resource (see e.g. (Babich, 1986)). This information could be collected and documented, and made available to the team to promote sharing within the team, and encourage more efficient collaboration by maintaining a collectively available source of information intended to empower the team members.

Configuration Audit

Audit is an important part of build and release, where all companies had some form of testing prior to a release/baseline to ensure that the product or service complies with the validation and compliance regulations (Daniels, 1985; Bendix and Ekman, 2007). In a DevOps context, audit can be viewed as something that takes place from day one, where every task is attached to a CI, requirement, or issue, which in turn is tested through unit testing and testing within the pipeline. Parts of these tests should reflect the configuration documentation, requirements, specifications, and change requests that should be met in order to establish a new

baseline for a release. Exactly which requirements, change requests or tasks should be included could likely be decided in, e.g. sprint planning, and by the end of the sprint a new baseline is established. Because a CI/CD pipeline is an integral part of DevOps, and CD implies the system is always in a releasable state, each point in time can be viewed as a potential baseline, and every commit triggering a new baseline, and with continuous deployment, this becomes reality as every commit is deployed to production.

Build management

As mentioned in all the interviews, build management was an essential part of the CI/CD pipeline, where build management handled the building and defining of a given system (Bendix and Ekman, 2007). As the build step is a part of the CI/CD pipeline, it must be automated in the DevOps context, which can be done using build management tools (e.g Jenkins). In addition, these tools should be used to specify a system(/dependencies) in a bill-of-material like document, where the BOM-documents could, among other things, include how the system was built, and what the build consist of. Also, by using build management tools, we provide the ability to derive different variants of a system.

SCM Plan

We refer the reader to section 4.5.2 for a description of our proposed usage of an SCM plan within the context of DevOps.

Release management

Release management is an important part of the DevOps activities build, deploy, and release, where it handles formal and informal releases to customers (Bendix and Ekman, 2007). Since release management includes everything from code integration to deployment, it should be automated with the help of tools. This produces reliable and repeatable processes in the DevOps context, where records of how the release was built, and what it consists of take place to ensure that the release can be recreated. However, to accomplish well produced and reliable records in DevOps there needs to be good traceability among components, modules, and dependencies. One needs to use version control to know which version of the different components, modules, and dependencies uses to create the system model, which in turn, becomes our Bill of material, which help us record things such as how the release was built, and what it consists of. Furthermore, when doing a release there needs to be some form of an audit, to verify that the correct system has been built. In DevOps, this audit can be seen as a part of the CI/CD pipeline, where every task is attached to a CI, requirement, or issue, which in turn is tested through unit testing and testing within the pipeline, effectively a continuous audit process.

Teamwork/Parallel work

From many of the characteristics of DevOps, such as culture and sharing, and what they entail, teamwork seems to be an important aspect to focus on. The SCM activity teamwork touches on, e.g. coordination, roles, and responsibilities (Appleton et al., 1998), which align with parts of the previously mentioned DevOps characteristics, in addition to collaboration

and trust, which is also important in DevOps (França et al., 2016). From the SCM interviews it was clear that teams should have collective ownership and responsibility for their work, which aligns well with DevOps characteristics. From the interviews most companies employed some variation of git flow or trunk-based development when coordinating parallel work within their version control tools. Most companies also left much of the decisions regarding exactly how they wanted to work to the teams themselves, which shows the team trust, and could be argued to both encourage teams to take responsibility, and to promote collaboration within the team in order to collectively agree on, e.g., a certain work flow.

Workspace management

From the interviews it was clear that git was the most used version control tool. Even so, in case other tools are used, there are potential workspace related problems which needs to be addressed, such as the three basic problems of shared data, simultaneous update, and double maintenance (Babich, 1986). Some version control tools enforces behavior which counters these problems, but we argue that being aware of them helps prevent them. A combination of informing about potential issues, as well as educating the users on the tools they use, e.g. how development in git works, local workspace and repository, what happens when certain commands are used etc.

Chapter 5
Discussion and Related Work

In this chapter we will discuss and reflect over challenges and limitations that we have encountered during the thesis work to critically review the results, and to discuss what we could have done better with the resource we had and did not have. To put the thesis into context, we will discuss and review related papers to our research. Furthermore, to strengthen and validate our results and guidelines, we will discuss the feedback that was received from the validators. Also, as we had limited resources, some of the potential work had to be continuously delimited, and in the section on future work we will provide a list with interesting discussions regarding these future work topics.

5.1 Reflections and limitations

In this section we will discuss and reflect over challenges encountered during the thesis work.

As the thesis work did not have unlimited resources, and had to be conducted under 20 weeks, limitations had to be set. For instance, if we did not have a deadline, we could spend resources on doing more interviews and literature studies, which might have added additional data. However, when the last interviews (both SCM and DevOps) and literature research were executed, we noticed that it did not provide a whole lot of new data, but rather information that could be used to confirm the already collected data and preliminary results. We could therefore conclude that we had collected enough data to see a conjunction between the different literature and interviews. Since we decided to interview a wide variety of companies, this provided a good width of information. There were clear differences between types of companies, and the largest differences could be identified, but to get a better understanding of these differences more research needs to be done.

Also, more time could have been spent on making the DevOps graph more “complete”, but since this was not our main focus, we had to postpone it for future work (see future work section).

Another thing that we might have wanted to improve was the interview questions that

were asked. In retrospect, the questions that were asked were in some cases too diffuse, which made the analysis part and the interpretation part of the answers much harder than necessary, which meant that we had to spend more resources on it than planned. Our idea with this approach was to ask more general questions to let the respondent answer more freely, instead of explicitly ask for the thing we were after. Even if this approach might have made it more complex than necessary, we gave the respondent more freedom, which led to interesting discussions and answers that we probably would not have received with more closed questions. However, since we had limited resources, this approach turned out to be both good and bad.

Furthermore, we had set a maximum time limit of one hour for each interview, but as all interviews became very interesting, time was easily forgotten. This lead to some of the interviews being longer than planned, which resulted in more resources than first expected had to be spent, especially in the analysis of the interviews, where one hour compared to one and a half could play a major difference in the time it took to analyze.

During the thesis work the partition on how many interviews we had under certain intervals could vary. This was something we could not control, because as the companies kindly wanted to participate, we tried to adapt to the their time schedules, which lead to about half of the DevOps interviews were done in a short time interval. Since we wanted to do the analysis part of the interviews while it was still fresh, we had to divide it between us. Even if this was a minor problem, where we still produced valuable analysis of the collected data, it probably would have been more optimal to divide the interviews under a longer range. This in turn would make the interviews less frequent, and let both of us analyse each interview, as we did on all the others.

Something that we realized when sending out the validation to the participation companies was that it might have been better to keep the companies interview answers separated in the report. This would make it easier for them to identify themselves in the report, and in turn analyse if they felt represented in our analysis on the interviews. Also, by separating the participating companies the readers would have been provided with more background and specifics of each company, and their answers. But this would also have cost, and in retrospect we would not want to reallocate time spent on the analysis regarding our research questions on this, as we believe our representation serves its purpose as is.

5.2 Related work

Since there was a lack of literature describing specifics on SCM implementation or usage in DevOps, we chose the paper *Software Configuration Management Practices for eXtreme Programming Teams* (Asklund et al., 2004), which roughly explains the same problem domain as ours, but instead of SCM in a DevOps context the authors evaluate SCM in the XP context. Even if it is different development methods, it is still very much relevant, as it relates to what this thesis explores. We then chose the papers *What is DevOps? A Systematic Mapping Study on Definitions and Practices* (Jabbari et al., 2016) and *Vad karaktäriserar DevOps?* (Mossberg et al., 2016), which tries to define and provide characteristics of DevOps, which is related to the first part of our thesis work. In this section we put our thesis work in relation to these three papers by providing a short summary and critique for each.

5.2.1 Software Configuration Management Practices for eXtreme Programming Teams

There exists a lot of literature describing the practices and philosophies of the programming method extreme programming (XP). However, the authors claim that there is very little literature and information about Software configuration management (SCM) in the XP context, if any, which creates the impression that SCM might not be needed in XP. This contradicts the authors view on SCM and XP, where they believe that no project can succeed without SCM. Therefore, they seek to clarify the implicit SCM process in XP, where they want to state SCM in XP explicitly, which will in turn help their students who work with XP projects with additional support, and also let other XP projects and SCM people benefit from their findings.

One of the main contributions from the authors were the design of SCM-related sub-practices, which covered the SCM requirements that were not satisfied by the standard XP-practices.

In addition, to accomplish the design of the SCM sub-practices the authors had to cover the general SCM activities needed in any project, and in turn identify to what extent these SCM activities cover XP practices, by both producing list of XP practices that implements SCM requirements, and list of SCM activities that is covered by XP practices.

The authors performed two study methods. The first study was done through literature, where they collected data about important SCM activities and XP practices (relevant to SCM). The second study was done with students working with the SCM-related sub-practices. This to further observe and investigate SCM sub-practices affection on students groups, both when the student groups were adapting after them and when they were not. However, as the authors has previous knowledge in XP and SCM, some facts and analysis could be based on own experience.

The studies were conducted step by step, where the authors started to use the literature studies to give the reader a brief introduction to the chosen SCM activities and XP practices. Based on the previous step and own experience, following analysis was done; XP seen from an SCM perspective, and SCM activities coverage on XP practices, which in turn was used to draw conclusions about SCM requirements that were met and not met.

Furthermore, based on the previous step, SCM-related sub-practices could be produced, which then was tested and validated on the students.

The paper presents that SCM is indeed present in XP. However, the authors express that there are SCM requirements that are not implemented by the XP practices. Nonetheless, they provide a solution for this by producing a list of given SCM sub-practices. This guarantees a complete development method in accordance with SCM requirements, which testing and validation of student XP projects confirms.

Discussion

In the paper the authors state that they only go in depth with XP practices that they consider relevant from an SCM perspective. However, there are no motivation behind why these XP practices is the most relevant. For instance, how did they chose these practices?, which

practices did they chose from?, and what was the deciding deciscions that contribute to the current five XP practices mentioned in the paper? If the author had mentioned all XP practices, and provided the reader with informative motivations behind the most relevant XP practices chosen, it would probably have increased the understanding of XP as a development method, and added more value in the form of why SCM is already present in XP.

This paper was important for the initial ideas of how to structure our thesis. It handles a similar situation of exploring SCM in a development method, and much of the methodology regarding how evaluation could be performed was inspired from its content.

5.2.2 What is DevOps? A Systematic Mapping Study on Definitions and Practices

The authors claim that software engineering often reuse existing terms for things that are not necessarily the same, which then results in misunderstandings regarding topics by using different definitions. The paper also claims that since the term "DevOps" (and its concepts) is relatively new, there is currently no common definition of what it means, and that the existing definitions only define parts of DevOps.

This lays the foundation for the author's task of conceptualizing DevOps by contributing three specific things (which are also their research questions). These are presented as comparing different definitions of DevOps found in literature, trying to identify practices and principles that appear to be related to DevOps, and lastly to perform a comparison between DevOps and other development methods.

The authors describe their overall methodology as being a systematic mapping study using a number of databases, based on their presented contributions, which mirrors the research questions presented. It is shown how literature was found, and where, and with the search strings they used providing general motivation for their choices, as well as how literature was filtered out to match their needs. Their filtering criteria shows that they focus on literature which is academic in nature, e.g. by excluding non-peer-reviewed literature.

Data is extracted from the compiled literature using a pilot-tested generalized form or checklist based on the data they wish to obtain in relation to their research questions.

Analysis of the selected literature is described as being a frequency analysis of the definitions of DevOps contained in the literature, through which the most common elements were extracted as part of a definition of DevOps. These common elements are then presented in a word cloud. The authors also describe how literature was analysed with regard to practices within DevOps, which is done by extracting practices which are explicitly mentioned in relation to DevOps. These are then categorized into different areas, and presented.

The authors conclude by proposing a definition of DevOps, based on their analysis, which contains a number key words identified as being representative of DevOps. With regard to the practices associated with DevOps, the steps mentioned in the analysis where practices were categorized, is presented as the answer to their second research question, as well as being used in the comparisons in the third research question. For the third research question, the authors show that there are (several kind of) relations between DevOps and agile software development, and also with cloud computing, ITIL, and quality assurance. The authors also mention that their results need to be validated, and that the value of different practices and combinations of them needs to be understood, and leaves this as future work.

Discussion

Although part of the stated goal was to only use peer-reviewed literature, probably to increase validity in their findings, it might limit the information and its current validity. The authors state that at least some of the literature they selected was based around the real life usage of DevOps, which is positive as it is not only speculation, but they also state in their section on validity threats that the definition of DevOps may evolve over time as new practices or components are added, which could imply that there could be value in also using non-peer-reviewed sources from people that actively work in DevOps or work with DevOps, to see if and/or how it is changing, connecting the academic point of view with the current industrial side of it more clearly.

As part of our initial characterization of DevOps, this paper was helpful to reference in order to get an overview of different subjects that might be interesting to explore. It proved to be a valuable resource, but severely lacking in depth and discussion, reducing it to mostly being used as a collection of topics to keep in mind throughout our characterization.

5.2.3 Vad karaktäriserar DevOps?

The authors claim, to deliver fast reliable software and to keeping up with the market speed, companies are increasingly trying to adapt to DevOps. However, despite the popularity, there exists a lack of unified definition of DevOps, where scientific literature about DevOps is inadequate, and scientific studies of it is absent. According to the authors, this implies that there is a need for empirical studies on DevOps in an organization, which in turn will contribute to an increased understanding of the development method, and be a valuable asset for further research on the phenomenon.

The main contribution from the authors was to identify what characterizes an organization working with DevOps, and what methods and characteristics are common between these organizations.

The overall methodology used by the authors is what the authors describe as a qualitative approach, seeking to explore the topic openly, consisting of interviews, followed by analysis.

The choice of interviews is motivated by the fact that they regard their research as more exploratory, as well as a way to achieve more width and some depth, as opposed to other methods which would achieve the opposite, citing the need for a descriptive result in their context (but also recognizing the need for both).

The authors describe their interviews as open, with a semi-structured approach, with questions ranging from open to guided. The interview data was analyzed iteratively and provided categorization of the interview data, which would then impact the following interviews, as a form of adaptation based on previous answers. The analysis of the interview data was performed by transcribing interviews and extracting key points by defining such points and mapping comments to them.

The paper describes the process of selecting interview candidates, and lists several characteristics, such as someone working in or with a group which identify as DevOps.

In terms of judging the validity of their results, the authors use the fact that if many people have the same view or understanding regarding something, it likely is close to "truth", and cite this as intersubjectivity. To achieve reliability, the authors also tried to stay neutral in interviews, only performing planned interviews, and recording the interviews.

In relation to the research question of "what are the characteristics of DevOps organizations?", the authors presents that CAMS had little support, with the motivation that Culture was far too wide and abstract, and Measuring was not a focus, and presents a new model for characterizing DevOps. This new model is described as focusing on insight and transparency throughout the lifecycle.

The authors also conclude that certain tasks are historically expected within a formal role, which, within DevOps, blurred the lines between roles as tasks and responsibility covered the whole development cycle. Their research also showed agile methodology was largely supportive, and that feedback usually was not a product of their own measuring, but rather information from other departments.

Discussion

The paper initially wants to answer the research question "what are the characteristics of DevOps organizations?", by performing interviews, and analyzing the interview data. They claim to not want to provide a definitive definition of DevOps through the paper but rather to present what they observe in DevOps organizations, but go on to provide a definition of DevOps through the framework CAMS (Culture, Automation, Measure, Sharing), and base their initial interview questions around it. Although the interview questions are continuously updated, their starting point is CAMS, and the author's analysis and categorization is (initially) in relation to CAMS, and later on their own framework, spawned from the initial categorization and subsequent analysis.

It seems counterproductive to base everything on a single framework, especially since it is apparent that some kind of literature study was performed, and a wide range of definitions, and therefore potential characteristics are presented and discussed, only to be discarded, at least initially. Even though the paper states that a new framework was developed over time, it is largely evaluated in relation to CAMS (and discussions surrounding literature), instead of perhaps the initial literature study, which might have provided a better evaluation and perhaps a better model, as the width of the discussed sources are likely more robust than a single point, as with CAMS.

Comparing the work done in this paper, and our characterization, it overall appears to be somewhat restrictive, and tightly tied to the specified framework CAMS. As such, we feel that this is a limiting factor, and its usefulness in our context (where we started with no assumptions regarding a characterization of DevOps) is therefore also limited.

5.3 Validation

In order to strengthen our results and confirm whether our guidelines could be applied in reality or not, a validation was sent out to eight companies. This section presents and discusses the main points from the received feedback from these companies.

In the validation we focused on the chapters *Characteristics of DevOps* and *SCM in a DevOps context*, although the validators were free to read and comment on the entire report if they chose to. The following two question were asked:

- Based on your participation in the interviews, do you feel represented in section 3.0 (DevOps) and/or 4.1 (SCM)?

- Do you think these results are applicable in reality (in your company)?
  - If not, please provide a short motivation why

Because of the way the questions are formulated, the absence of critique or comments in the received feedback is interpreted as agreement. That being said, of the responses we have received, everyone has felt represented in section 3.0 and 4.1. Currently, no one has commented on whether the results are applicable in reality, nor have they provided comments on them.

A validator who participated in interviews on DevOps noted our usage of the term "DevOps team", asking how "DevOps team" compares to the DevOps paradigm. This is a valid point, and a reiteration of parts of the introduction to this thesis, where we discuss the fact that there is no agreement on whether DevOps is something you *are*, or something you *do*, as well as the lack of operational definition of DevOps, making it difficult to discuss a DevOps paradigm. Within the context of our thesis, a "DevOps team" is a team who's development methodology, principles, and practices, overlaps with our characterization of DevOps. This validator also commented on SCM plans, and it not being needed in DevOps, with the motivation that a team should work the way that fits them best, and that an SCM plan likely is not up to date, or relevant to all teams. This agrees with our own analysis and alternative approach to SCM plans in chapter 4.

Overall the respondents expressed agreement in being represented in the interview discussions. They were also positive towards the application of the results, or had the opinion that it helped confirm important items.

5.4 Future work

The path to our results for this thesis work has been far from straight, which has led us to encountering many interesting things. However, due to time constraints, we have not been able to examine all parts, and have been forced to postpone it for future work. This section provides prospective points for future work of this thesis.

1. For the future work, there could be more work spent on making the graphs more "complete", where one could e.g. look more into other connections between the characteristics/activities, i.e. other types of dependencies (edges) between the characteristics/activities and possibly categorize them, and what type of abstraction levels the characteristics could be divided into, as it is clear that there are different levels.

2. As time was limited, it did not allow for us to test our guidelines in practice, and so it had to be left for future work. The tests of the guidelines could (e.g.) be done with case studies, where differences in usage based on type of company could be looked into, and where/how to begin a DevOps transformation for a given company type, or where/how a startup company would/could apply these based on their needs. Also, another interesting thing to investigate could be what kind of difficulties there are for established (e.g. waterfall) companies with strict SCM processes moving towards more agile variations, when moving towards DevOps, and conversely for newer companies or startups wanting to use SCM.

3. More future work could be done in the form of exploring more DevOps characteristics, since the characterization of DevOps was not the main focus of this thesis. This means that more time could have been spent on, for instance, trying to identify more characteristics, do more research about the current characteristics, and maybe change the current ones, and/or explore the validity of the ones we have identified.

4. During some of the interviews we encountered the word "cost", which referred to the cost of doing/going DevOps, where there is a need for future investigation if the cost is worth it, and what the costs are, both in relation to the profitability of doing so, and also where costs might be, i.e. what "things" in DevOps that are possibly going to be a cost.

5. Lastly, another topic that could be interesting to look into is if the graph strategy we used could be abstracted and generalized, perhaps suited in combination with point 1 in this section, and then be used on other software development methods. This could potentially be used to compare different development methods, and see similarities and differences between them, which might be useful when switching between methods or analysing what might be suitable based on one's needs, as well as providing a plan for implementation, as the dependencies are immediately available.

Chapter 6
Conclusion

In conclusion it can be seen that all SCM activities were deemed as needed within DevOps (which answers research question RQ1a), either because they were already present, or because of the value they bring, with the exception of SCM plans in their current form, which were regarded as too specific and rigid in an agile environment.

Based on our previous analysis, we introduced an alternative to SCM plans (research question RQ1b), which instead includes the overall DevOps process one has adopted, as well as SCM activities in a common resource, in order to counteract a separation of the two, and to promote understanding of them both, why they are important, how to do them, and to present them as being part of the same overall process.

Lastly to answer the research question RQ1c, we provide generalizable guidelines for how SCM activities can be done in DevOps, based on how SCM activities mapped to DevOps characteristics. These guidelines provide valuable ideas regarding both where a given SCM activity fits in within DevOps, and where a subset of DevOps fits into SCM. The guidelines also show how the definitions of SCM activities can be interpreted to allow for a more agile usage within DevOps, based on one's needs, which could help both newer startups as well as larger "waterfall" companies to either include SCM, move towards DevOps, or both.

We would also like to highlight parts of the section on future work discussing our graph of DevOps characteristics and the combined graph including the SCM activities, as these were instrumental in understanding the interconnectedness of the identified DevOps characteristics, and being able to map these to SCM in a structured way, making them, in our opinion, a valid result on their own.

Bibliography

Appleton, B., Berczuk, S. P., Cabrera, R., and Orenstein, R. (1998). Branching patterns for parallel software development. [internet]. In *Streamed Lines*. [cited 2019 Oct 17]. Available from: http://www.bradapp.com/acme/branching/branch-forces.html.

Asklund, U., Bendix, L., and Ekman, T. (2004). Software configuration management practices for extreme programming teams. In *Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques NWPER'2004*, Turku, Finland.

Auerbach, A. (2017). Why devops still needs release management [internet]. CM Crossroads. [updated 2017 Nov 15; cited 2019 Oct 17]. Available from: https://www.cmcrossroads.com/article/why-devops-still-needs-release-management.

Babich, W. A. (1986). *Software Configuration Management – Coordination for Team Productivity*. Addison-Wesley.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). The agile manifesto. [internet]. Available from: http://agilemanifesto.org/.

Bendix, L. (2019). A short introduction to software configuration management [internet]. [cited 2019 Oct 24]. Available from: http://fileadmin.cs.lth.se/cs/Personal/Lars_Bendix/Teaching/1-ECTS-SCM/90min/Oslo-19/LNSCM.pdf.

Bendix, L. and Ekman, T. (2007). Software configuration management in agile development. In *Agile Software Development Quality Assurance*. Information Science Reference.

Bendix, L. and Pendleton, C. (2019). Scm in a devops context [powerpoint presentation]. Copenhagen. Available from: http://fileadmin.cs.lth.se/cs//Personal/Lars_Bendix/Research/SCMnDevOps/Meetup-ITU-v4-4pp.pdf.

C&A (n.d.). Configuration control [internet]. Chambers & Associates Pty Ltd. [cited 2019 Oct 17]. Available from: http://www.chambers.com.au/glossary/configuration_control.php.

Casey, K. (2019). Agile vs. devops: What's the difference? [internet]. The Enterprise Project. [updated 2019 Jan 3; cited 2019 Oct 17]. Available from: https://enterprisersproject.com/article/2019/1/agile-vs-devops-whats-difference.

Daniels, M. A. (1985). *Principles of Configuration Management*. Advanced Applications Consultants, Inc.

Daskalopoulos, A. (2019). Test everywhere: A journey into devops and continuous testing [internet]. CM Crossroads. [updated 2019 Jul 15; cited 2019 Oct 17]. Available from: https://www.cmcrossroads.com/article/test-everywhere-journey-devops-and-continuous-testing.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. pages 94–100. IEEE Software, 33 edition.

Erich, F., Amrit, C., and Daneva, M. (2014). Report: Devops literature review.

Feiler, P. H. (1991). *Configuration Management Models in Commercial Environments*. Software Engineering Institute.

Forsgren, N., Smith, D., Humble, J., and Frazelle, J. (2019). Accelerate state of devops 2019 [internet]. [2019; cited 2019 Nov 13]. Available from: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf.

Fournier, C. (2019). Engineer's guide to identifying and coping with path dependence [youtube]. [2019; cited 2019 Nov 14]. CoDe-Conf 2019, Copenhagen. Available from: https://youtu.be/6O75onO1BIQ?list=PLuvRKxeqrv4It4SWS84qW66wIfobtpSDu.

França, B. B., Jeronimo, H., and Travassos, G. H. (2016). *Characterizing DevOps by Hearing Multiple Voices*. SBES.

Guckenheimer, S. (2017). What is infrastructure as code? [internet]. Microsoft. [updated 2018 Oct 24; cited 2019 Oct 17]. Available from: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code.

Jabbari, R., Ali, N., Petersen, K., and Tanveer, B. (2016). *What is DevOps? A Systematic Mapping Study on Denitions and Practices*. Association for Computing Machinery, New York, NY, USA.

Kelly, M. V. (1996). What does and does not need controlling. In *Configuration Management - The Changing Image*, pages 55–66. McGraw-Hill Book Company.

Kornilova, I. (2017). Devops is a culture, not a role! [internet]. Medium. [updated 2017 Apr 28; cited 2019 Oct 17]. Available from: https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0.

Leon, A. (2014). *A Guide to Software Configuration Management*, chapter 11. Artech House Inc.

Lwakatare, L. E., Kuvaja, P., and Oivo, M. (2015). Dimensions of DevOps. Springer International Publishing, Switzerland, 2015.

Mann, A. (2019). Devops: Shift left to reduce failure [internet]. [2019; cited 2019 Nov 14]. CoDe-Conf 2019, Copenhagen, 28 Oct 2019. Available from: https://www.youtube.com/watch?v=WYU2XZ1nCpk&t=4s.

Mossberg, S., Friberg, G., and Joelson-Tui, J. (2016). Vad karaktäriserar devops? Student Paper. Available from: http://lup.lub.lu.se/student-papers/record/8881167.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley. Seventeenth printing, May 2013. ISBN: 0-321-15078-3.

Roche, J. (2013). Adopting devops practices in quality assurance. In *Commun. ACM*. 56 edition.

Rushgrove, G. (2016). Devops: A culture of sharing [internet]. [2019; cited 2019 Nov 18]. Puppet. Available from: https://puppet.com/resources/video/devops-culture-sharing/.

Schrag, D. (2016). Devops: Shift left to reduce failure [internet]. [2016; cited 2019 Nov 14] Available from: https://devops.com/devops-shift-left-avoid-failure/.

Sharma, S. (2014). *DevOps For Dummies, IBM Limited Edition*. John Wiley & Sons, Inc., Hoboken, New Jersey.

Smeds, J., Nybom, K., and Porres, I. (2015). DevOps: A Definition and Perceived Adoption Impediments. Abo Akademi University, Finland, 2015.

Ståhl, D., Mårtensson, T., and Bosch, J. (2017). Continuous practices and devops: beyond the buzz, what does it all mean? In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria. IEEE, IEEE.

TechInsights (2013). Techinsights report: What smart businesses know about devops [internet]. TechInsigths. [cited 2019 Oct 24]. TechInsights Report. Available from: https://www.logicworks.com/wp-content/uploads/2016/01/techinsights-report-what-smart-businesses-know-about-devops.pdf.

Vega, C. (2019). Ikea's journey into digitalisation. [youtube]. [2019; cited 2019 Nov 19]. CoDe-Conf 2019, Copenhagen, 28 Oct 2019. Available from: https://www.youtube.com/watch?v=Zukki9fI2yw&t=828s.

Yigal, A. (2016). How devops is killing qa [internet]. [2016 Jun 7; cited 2019 Nov 6]. Available from: https://devops.com/devops-killed-qa/.

Appendices

Appendix A
DevOps interview template

1. Tell me a bit about yourself
    (a) Background
    (b) Role
    (c) Team
2. What type of company do you work for?
    (a) Type of company (traditional, agile, startup etc.)
3. Why do you think you are DevOps and not something else?
    (a) How have you achieved DevOps in your team/organization?
    (b) Which problems do you want to solve using DevOps?
        i. How do you try to achieve that?
    (c) There are many development methodologies, how come DevOps was chosen?
    (d) How has your team developed after adapting to DevOps?
        i. Did something in your existing process stop working?
        ii. Did something in DevOps not work?
4. How did DevOps affect the release cycle?
    (a) For example, how has your time to deployment increased/deployment frequency changed?
        i. How did you improve this with DevOps? (if improvement)
        ii. How did your process change?
        iii. Did you remove something that didn't work with DevOps?

A. How did it affect you process(es)?

5. If something goes wrong along the way, how do you solve it? (Maybe give example on scenarios)
   (a) Can you, wherever you are in the process, know if something goes wrong?
      i. How is this achieved?
      ii. How do you handle these changes?
6. How do your change process work?
   (a) How do you handle change requests?
7. How do you collaborate within the team/How is teamwork performed?
   (a) Do you have fixed roles in the team?
      i. How is information shared within the team?
8. How do you communicate within the team?
   (a) Is this something that has been improved because of DevOps?
   (b) How is information shared within the team?
9. What was the hardest part to change when becoming a DevOps company/team?

Appendix B
SCM interview template

1. Tell me a bit about yourself (if it is a first time interview)
    (a) Background
    (b) Main role
    (c) Team
2. What type of company do you work for? (if it is a first time interview)
    (a) Type of company (traditional, agile, startup etc.)
3. How/Where do you use Configuration Identification in your development process?
    (a) What do you see as a CI?
    (b) How do you handle these CIs?
4. How do your process for controlling changes work?
    (a) CCB?
    (b) Where/when?
5. Can we get information about CI:s wherever we are in the process?
    (a) CMDB?
6. How is the process for baselining structured?
    (a) Physical/functional requirements?
7. Do you have a process for specifying a system and how to build it?
    (a) Is the configuration version controlled?

   (b) Variants?

8. How do you collaborate within the team/How is teamwork performed? (if it is a first time interview)
9. How do you manage your workspaces?
   (a) Branching strategies (?)
10. What is covered by your version control?
   (a) Code, monitor data, configurations, etc. (?)
   (b) What is not covered?
11. Can one track changes from the origin of the change to the implemented source code?
   (a) Traceability throughout project/product
12. Do you plan your SCM activities, and document them?
   (a) SCM Plan
13. How do you release?
14. How is a change triggered?
   (a) Change request via forms, or something similar?