Software development using DevOps tools and CD pipelines, A case study

Oskari Jokinen

Master's Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, February 25, 2020

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Oskari Jokinen

Työn nimi — Arbetets titel — Title

Software development using DevOps tools and CD pipelines, A case study

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's Thesis	February 25, 2020	50

Tiivistelmä — Referat — Abstract

The objective of this study is to get insight on the usage of automated build and deployment pipelines by software development teams at Avaintec Oy in the context of DevOps methodologies. Automated deployments are used widely in today's software development and they allow for complex installation operations and testing to be done in a time-saving manner as less manual work is required.

While there is a lot of research in the area of DevOps and automation pipelines, standardised ways of working and best practices are still vague in many ways. Studying the effects of DevOps and the use of the automation pipelines in the scope of the development team is important to establish practices and ways of working that can best support the work of the developers and also be cost effective for the companies.

The research methods of this study are a literature review of the most recent literature on using automation in software development, and a case study of the process of taking DevOps methodologies and a Continuous Deployment pipeline into use at Avaintec Oy. The literature is analysed with the help of a theme matrix. The case study was conducted by interviewing selected members of development teams at Avaintec. The interviews were recorded and transcribed, and then analysed with a theme matrix. The output from the literature review is reflected to the output from the interviews in the case study section.

The conclusions from this study indicate that development teams find DevOps and DevOps pipelines useful, despite the learning curve in the new tools and methodologies and the amount of initial setup work. The evolution of an automated build and deployment pipeline should be continuous and pipelines should be built incrementally, both of which were true for the studied case. The engagement of developers to the Operations side is a challenge present in both literature and the studied case, as developers at times are not that familiar with the pipeline. Apart from the pipeline, products should also be developed to be suitable for DevOps and automatic deployments, microservices seem to provide advantage in this.

ACM Computing Classification System (CCS):
Software and its engineering → Software notations and tools
Software and its engineering → Software creation and management

Avainsanat — Nyckelord — Keywords
DevOps, Continuous Integration, Continuous Delivery, Continuous Deployment, Deployment Pipeline

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

As software product development faces more demand from the customers and getting a market share is increasingly harder, software companies must find ways to increase the efficiency of the development work. Traditionally software developers have been concentrating on writing the software source code and then the company's operations team has built and installed the product onto some platform. This traditional method is however time-consuming and the time from feature idea to production can take a long time.

DevOps introduces the idea of not having separate Development and Operations -teams but a multi-skilled team working together with an automation pipeline to make it possible to deploy faster and with less effort. When put together correctly, the team and the DevOps tool pipeline should be able to produce more in less time.

In this thesis the view of the cross-skilled team is reviewed by looking at literature around the matter and also by conducting a case study of the start of usage of DevOps tools in Avaintec Oy, where new deployment and development strategies as well as automation tools were taken into use in the recent years. The focus of this study is in the development team and the tools they are using for the development and automation process.

For conducting a research in this matter, the following research questions were formed;

1. How do software development teams experience the usage of DevOps pipelines?

2. How does a DevOps tool chain evolve during its lifespan?

3. How should DevOps pipelines be used for them to be helpful for the development team's work?

In the following chapters, the Concepts of DevOps and the related tooling is introduced. Then recent literature is researched for relevant views on the matter of DevOps from the development teams perspective. A Case study conducted with interviews of a multi-skilled team is presented and the output from the interviews is reflected to the findings from the literature. In the discussion section the threats to the validity of this thesis and possibilities for future research in the field are presented. Finally in the conclusions section the findings of this study are presented.

The goal of the study is to understand, with the help of the research questions, the view of the Avaintec development team towards the DevOps practices and tools and how they relate to the findings from recent literature.

2 Background

DevOps as a term comprises of the words "Development" and Operations". Traditionally in a software development scope Developers write the source code and then people responsible of installations install it to some platform. In the DevOps philosophy these functions are a joint venture of the developers and operations teams [Ebert et al., 2016]. The goal of this way of working is to reduce the time it takes to commit changes from the beginning of the development cycle to a production system[Balalaie et al., 2016]. In the next few chapters the different aspects of DevOps and DevOps related themes are introduced to give backgrount to the study.

2.1 Technological DevOps

When DevOps is mentioned the first thing that usually comes into mind are the tools required. To make DevOps possible or feasible, automation tools are often used to get results with less repetitive tasks[Ebert et al., 2016]. This is valid especially for the version control, compilation, build, testing, deployment and delivery phases of software development as they often have the same phases, somewhat independently of the new additions to the actual software source code.

The technological advantages of DevOps automation tools comprise of shorter release cycles for the software, Continuously having a working or releasable version available of the software [Shahin et al., 2017a] and the ability to rely on the delivery process. In essence, as human factors are eliminated the deployment and delivery process ought to be faster and the deployments less likely to fail or have bugs or configuration errors comparing to a manual installation.

2.2 DevOps Culture

A very important part of DevOps, probably even more significant than the technology part is the human part in DevOps. In many software companies the delivery process of software is controlled by the management [Shahin et al., 2017b] and thus automating the software delivery process may require changes to the acceptance policies.

Another important aspect in the culture domain is the role of Developers and the Operations people or system administrators. These roles are becoming more vague in their distinction when talking about DevOps, as the Developers are given tasks from the operations side and also the Operations people need to understand more of the product that is being developed. In some circumstances this may cause change resistance in organizations[Shahin et al., 2017a].

As DevOps has both, Development and Operations, it's good to remember that if those two lines of practice work as separate units, it is highly likely that it's possible to produce automated deployment pipelines and technologically advanced delivery processes. However, if the Development and evolution of these pipelines is not a joint venture of the Development and Operations personnel, it is not DevOps.

2.3 Automation in software development

In the modern world of software development, frequent updates to software, continuous feedback to the developers of their work and fast delivery times are of essence[Shahin et al., 2017a]. Automating the software delivery process is one of the tools trying to provide to these needs. When using an automated deployment pipeline, the developers will get feedback of their code regularly, bugs can at times be found before the code is even committed to the shared repository and minor changes to the software don't need to wait for the next major release, but can be done faster [Shahin et al., 2017a]. The different

concepts and stages of development automation often get confused with each other. The naming of these stages is rather alike within the automation types, thus they get easily mixed up. In the next chapters the different types of software deployment automation are introduced in the form that they will be referred to in this study.

2.4 Continuous Integration

The objectives of Continuous Integration (CI) are to first gather the code from the different members of the development team into a source code repository where the code is merged and changes can be version controlled [Ebert et al., 2016]. In the source code repository different development versions and a production version of the software can be stored, thus making it possible to develop the software without touching the working versions.

Second step in continuous integration is to build the source code into a working software and then run automated tests on the code. This way it's always known that there is a working version of the source code available at all times which can be deployed to an installation platform. As the integration is ongoing and automatic, conflicts in code can usually be quickly found and resolved or reverted [Shahin et al., 2017a].

2.5 Continuous Delivery

Continuous DElivery (CDE) is a natural next step for continuous integration. Continuous integration is considered a mandatory part of continuous Delivery, thus Continuous Delivery consists of Continuous Integration and the additional steps from Continuous Delivery[Chen, 2015]. In continuous Delivery the desired version of the software is automatically installed to a desired platform, for example to a web server[Shahin et al., 2017a]. Usually this platform is a production-like environment where it can be verified that the software stays in a state that can be deployed to production at any time.

It is typical that a Continuous Delivery is performed onto test system every time that there has been a change in source code[Chen, 2015]. Different deployment platforms can exist for different source code versions or branches, for example a development branch can be continuously deploying on to a "CI" environment which is a lighter version of the production environment, and then a Staging or pre-release branch can be deploying automatically on an exact or as-close-as-possible replica of the production environment.

In Continuous delivery the deployment to production can also be automated, but it requires manual interaction at some state, thus distinguishing it from Continuous Deployment[Chen, 2015]. The benefit of having a continuous delivery pipeline to a environments is that the installation has been ran the same way multiple times and is thus tested to certainly function correctly.

2.6 Continuous Deployment

Continuous integration (CI) and Continuous Delivery (CDE) are the base for Continuous Deployment[Humble and Farley, 2010]. Continuous deployment is the combination of the aforementioned phases and the most sophisticated version of an automated delivery pipeline.

In continuous deployment the product is deployed fully automatically all the way to the production environment without any human interaction on the way [Shahin et al., 2017a]. Thus all the changes that are done to the source code are automatically deployed. In many domains the automated production delivery is not possible due to customer restrictions or company acceptance policies. One way of looking at the difference is to consider Continuous Delivery a pull-based deployment as the changes need to be manually taken into production, as Continuous deployment is a push-based approach as the changes are fully automatically pushed into production.

3 Research methods

This thesis is based on a Literature review and a Case study which is implemented using themed interviews. The methods of these research types are introduced in the following chapters. The Literature review method is based on Webster, J. & Watson, R., 2002. Analyzing the past to prepare for the future: Writing a literature review [Webster and Watson, 2002]. The Case study method is based on Runeson, P. & Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering [Runeson and Höst, 2008] and Charmaz, 2006. Constructing grounded theory - A practical guide [Charmaz, 2006]

3.1 Literature review

A Literature review was conducted to research the most recent material on the topic. To select a range of recent papers on the field, two recent systematic literature reviews related to Continuous Integration, Continuous Delivery and Continuous Deployment tools were selected as starting points for the literature search. This was done by searching in Google scholar with the search string "Continuous Integration Continuous Delivery Continuous Deployment Tools "Systematic Literature Review", and limiting the results to articles from 2017 or later. Of these results the most relevant SLR:s with most citations at the time were selected. The selected literature reviews were:

- Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, Shahin, M., Ali Babar, M., Zhu, L. 2017 [Shahin et al., 2017a]
- Problems, causes and solutions when adopting continuous delivery—A systematic literature review, Laukkanen, E., Itkonen, J., Lassenius, C. 2017 [Laukkanen et al., 2017a]

A forward reference lookup was done from these articles to find the most recent articles that had referred to these two SLR:s that are in the core of the topic of this study. A total of 35 articles referring to these literature reviews were found when the search was conducted in December 2019. The citing articles were listed in the original publication sources of the articles, IEEE Xplore Digital Library for [Shahin et al., 2017a] and Elsevier Scopus for [Laukkanen et al., 2017a]. These articles were selected for further inspection. Of those 35 articles 4 were overlapping, in essence referring to both of the aforementioned literature reviews.

A preliminary selection of articles was done by selecting only the articles that were published conference papers and not literature reviews. At this stage two papers were discarded for being proceedings talks, one for being an systematic literature review and three for not being available at all or not being available in the English language. Of these the articles that had any of the following keywords specified were selected for reading:

- Continuous Integrati -on -ons
- Continuous Deliver -y -ies
- Continuous Deployment

This resulted in two articles being narrowed out of the scope of the literature review.

After narrowing down the articles, the articles were read and their validity for the study was evaluated. Requirements for suitability was that they must have references to usage or selection of DevOps Tools, to the evolution of a DevOps pipeline or to the process of adopting a DevOps pipeline into use in a software development project. As a result there were 11 articles left to be studied.

The articles were then systematically analyzed by extracting the central concepts related to the research questions of this study from them. A mapping

was done on the concepts The literature was analyzed with focus especially on how the use of DevOps tools has been experienced in development teams, how it has affected the work of the team and team dynamics and how the DevOps toolchains and pipelines have evolved during the time that they have been in use.

3.2 Selected papers for the literature review

The selected papers contain a variety of recent articles that discuss the topic of DevOps, Continuous delivery, Continuous deployment and DevOps tools and usage of them in the scope of the development team. Many papers that were preliminary selected from the forward reference search were discarded due to some form of irrelevance for the study. The irrelevance of an article to the study was defined by the article not having any input on the matter of DevOps philosophy or tool usage in the scope of a software development team.

The selected and discarded papers are presented in Table 1 with the reasons for Discarding the paper including irrelevance to the study, unavailability, missing keywords or discarding by article type. The selected articles are numbered S1-S11 and are referred to later using this number reference and the bibliography reference.

3.3 Case study

A qualitative case study was conducted on the usage of DevOps pipelines in the case of Avaintec Oy with their newer cloud products and their development process. The research was conducted by themed interviews. The structure and the questions in the team interview can be found in appendix A. Selection of the people to be interviewed was done by selecting the key persons involved in the development process. As the development teams have varied and people have worked cross-product the selection did not need to have a separate

Table 1: Article selection matrix

Unrelated to study	A Continuous Delivery Pipeline for EA Model Evolution Hacks, S., Steffens, A., Hansen, P., Rajashekar, N. 2019
Unrelated to study	A Review of Source Code Management Tools for Continuous Software Development Uzunbayir, S., Kurtel, K. 2018
S1	Adoption issues in DevOps from the perspective of continuous delivery pipeline Zulfahmi Toh, M., Sahibuddin, S., Mahrin, M.N. 2019
S11	An empirical study of architecting for continuous delivery and deployment Shahin, M., Zahedi, M., Babar, M.A., Zhu, L. 2019
Unrelated to study	Atom-Task Precondition Technique to Optimize Large Scale GUI Testing Time based on Parallel ... Wongkampoo, S., Kiattisin, S. 2018
S5	Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust ... Garg, S., Garg, S. 2019
S2	Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges Shahin, M. et.al 2017
Unrelated to study	Collecting data from continuous practices: An infrastructure to support team development Nogueira, A.F. et.al 2019
S10	Comparison of release engineering practices in a large mature company and a startup Laukkanen, E. et.al 2018
Proceedings talk	Continuous delivery at scale: Challenges and opportunities Chen, L. 2018
S3	Continuous Delivery: Overcoming adoption challenges Chen, L. 2017
Unrelated to study	Continuous integrated team learning Soares, L.P., Ferrão, R. 2018
Unrelated to study	Continuous integration and continuous delivery pipeline automation for agile software project management Arachchi, Perera, I. 2018
Not available	Continuous integration for android application development and training Hung, P.D., Giang, D.T. 2019
Language availability	Continuous integration in distributed applied software packages Feoktistov, A.G., Gorsky, S.A., Sidorov, I.A., Tchernykh, A. 2019
SLR	Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review Mascheroni, M.A. et.al 2018
S4	Designing a next-generation continuous software delivery system: Concepts and architecture Steffens, A., Lichter, H., Döring, J.S. 2018
Unrelated to study	Developing cross-organisational service-based software systems through decentralised interface-oriented ... Almalki, J. et.al 2018
Unrelated to study	Devops round-trip engineering: Traceability from dev to ops and back again Jiménez, M. et.al 2019
No keyword match	Efficient static checking of library updates Foo, D., Chua, H., Yeo, J., Ang, M.Y., Sharma, A. 2018
No keyword match	Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps ... Soenen, T. et.al 2018
Unrelated to study	Key Affordances of Platform-as-a-Service: Self-Organization and Continuous Feedback Krancher, O., Luther, P., Jost, M. 2018
Unrelated to study	Managing quality assurance challenges of Devops through analytics Ahmad Ibrahim, M.M., Syed-Mohamad, S.M., Husin, M.H. 2019
Not available	Maximizing the efficiency of automotive software development environment using open source technologies Niaetin, S. et.al 2018
Unrelated to study	Method of development and deployment of reconfigurable FPGA-based projects in cloud infrastructure Kulanov, V. et.al 2018
Unrelated to study	On the Use of Automated Log Clustering to Support Effort Reduction in Continuous Engineering Rosenberg, C.M., Moonen, L. 2018
S8	One size does not fit all: An empirical study of containerized continuous deployment workflows Zhang, Y. et.al 2018
S7	Practitioners' eye on continuous software engineering: An interview study Johanssen, J.O., Kleebaum, A., Paech, B., Bruegge, B. 2018
Unrelated to study	Software analytics in continuous delivery: A case study on success factors Huijgens, H. et.al 2018
Proceedings talk	Strategy for continuous testing in iDevOps Zimmerer, P. 2018
S9	Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat's pocket code Luhana, K.K. et.al 2018
Unrelated to study	Towards a continuous feedback loop for service-oriented environments Martin, K., Omer, U., Florian, M. 2018
S6	Towards continuous delivery by reducing the feature freeze period: A case study Laukkanen, E. et.al 2017
Unrelated to study	Utilising CI environment for efficient and effective testing of NFRs Yu, L., Alégroth, E., Chatzipetrou, P., Gorschek, T. 2020
Unrelated to study	We're doing it live: A multi-method empirical study on continuous experimentation Schermann, G. et.al 2018

interviewee set from each product team. Instead, a software architect, a technical product owner, an UI/UX designer , a System administrator and a QA engineer were interviewed. The people interviewed (N = 5) were selected from a wide range of seniority in the company, and also the goal was to interview people who also have experience in working without the DevOps pipelines in earlier projects. The basic details about the interviewees can be seen in table 2.

The interviews were transcribed and an initial and focused level of coding was applied to find recurring points and views on the matter given by different

Table 2: Interviewee details

ID	Position	Years on industry	Products worked with
1	QA/Tester	15	Signhero, Avainpay, Datachief, Legacy products
2	System Administrator	3	Signhero, Avainpay
3	UI developer	10	Signhero, Avainpay, Datachief, Legacy products
4	Software engineer	5	Signhero, Prews, Cloudlace
5	Technical product owner	5	Signhero

interviewees. The coding was done with reference to [Charmaz, 2006] The outputs from the interviews were then reflected to the output from the Literature review to get a view on how the case findings at Avaintec Oy resonates with the findings in recent studies from the field. The validity of the findings was reinforced by presenting them to the interviewees, thus conducting member checking.

3.4 The case: Avaintec Oy

Avaintec Oy is a software company founded in 1997. Avaintec's product portfolio has included Digital signatures, Digital archiving, Digital forms, AI and Data Analytics and also mobile payments. The largest customers for these services have been in the healthcare and Banking sectors. Today's portfolio of products at Avaintec is focused on eSigning (Digital signatures) and AI / Data analytics solutions. Due to the long history in the field of software development, Avaintec has a broad experience of product development and delivery. During the recent years and in the introduction of the newest products to the portfolio, the build and deployment of these new products to the test systems as well as some of the production deliveries were automated. This was done in the search of streamlining the development process and avoiding the issues that have been experienced with the current product portfolio whilst doing installations or version updates. Also the nature of the new products being SaaS services, for example the eSigning solution "SignHero", a deployment solution able to support a scalable deployment environment and frequent service updates was a requirement.

4 The context

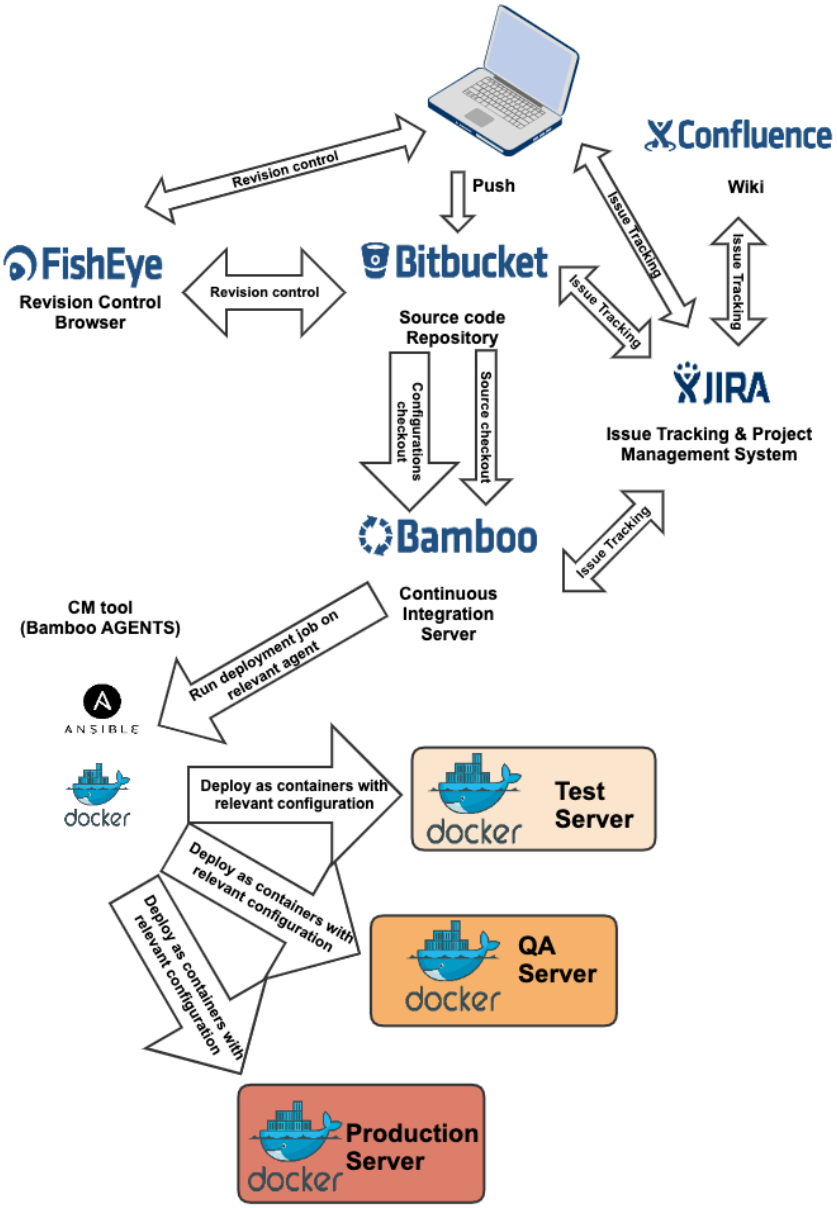
In this chapter the setup for this study is presented as the tools selected for the DevOps pipeline used at Avaintec are introduced. The tools introduced here are the parts of the DevOps pipeline that make the different phases of automation mentioned earlier possible.

4.1 Selected tool set

The tool set described below was selected for use at Avaintec after considering multiple factors. The pipeline needed to be versatile enough to be able to handle different kinds of operations related to deployments, it had to be relatively straight-forward to configure and also suitable for usage with the tools already in use in the company. Integration possibilities to existing systems were important. Thus the pipeline was built around the Atlassian suite as Confluence wiki and Jira issue tracking software had already been used by Avaintec for years.

A visual representation of the tool set is presented in Figure 1

Figure 1: A schematic of the Avaintec deployment pipeline



4.1.1 Issue tracking

Atlassian Jira issue tracking is considered the beginning of each new feature and release of the software. In Jira it's possible to create feature tickets

and break larger features down to smaller implementations, which are then produced into source code.

The issues in Jira are often related to branches in code in the version control system, thus making it possible to track the progress of the issue and code related to it. Jira is also able to integrate with the Build and deployment automation to keep track of the builds related to certain issues and vice versa.

4.1.2 Source code repository

The Atlassian Bitbucket serves as a source code repository for the system. Bitbucket is based on Git version control, which keeps a log of every change made to the source repository, which allows you to revert to a previous state in the code. Thus, different versions are not preserved as a whole, but only differences between different versions.

Bitbucket integrates seamlessly with other products in the Atlassian family, such as Jira, which is able to handle code changes made to Bitbucket directly on its tickets. In this case, changes to the code can be directly reviewed from the ticket which makes managing the issues and related code changes easier.

In the tool set the different sets of configurations are also stored in Bitbucket. Ansible, Gradle and the Docker containers on which most of the components are ran require a set of configuration files, all of which are stored in Bitbucket for collaborative editing and version control. In essence, the configurations are handled similarly as any piece of source code.

4.1.3 Revision control

Atlassian Fisheye serves as the revision control software for our pipeline. Fisheye makes it possible to examine the code with search and difference examination capabilities, and being accompanied by Crucible it's also possible

to make collaborative code reviews using the tool. Some of the functionalities in Fisheye are also available natively in Bitbucket so the need for Fisheye in the pipeline is under consideration.

4.1.4 Build and deployment automation

Atlassian Bamboo is used as the build deployment automation tool in the pipeline. In Bamboo, it's possible to configure the source code you want to retrieve and its version (or branch), from which you start working on a publishable package. Bamboo includes many interfaces for using different types of unit tests, different compilers, and different configuration tools. In addition, Bamboo is able to execute several such tasks simultaneously with several agents, which speeds up the execution of the tasks. The build and deployment phases are separated in Avaintec's solution, thus each product has their own build plan which can be used for multiple branches. The results of those branches can then use the environment-specific deployment plans to be deployed onto a platform. All of these stages can be triggered via automation either by keeping track of changes in selected source code repository branches, or by a time controlled trigger.

4.1.5 Configuration injection

Ansible is used alongside Bamboo in the configuration part of the deployment. Ansible playbooks are collections of commands that are used to configure the installation platform for the deployment. The playbooks are written in an easily understandable YAML syntax. The playbooks are stored in Bitbucket, which allows them to be collaboratively modified and the configuration versions to be controlled and rolled back if necessary.

4.1.6 Build automation

Gradle is used as the build automation tool in the pipeline. Gradle is capable of building packages of Java, JavaScript, C++, Android and Python sources. The build configurations are declared for Gradle using a Groovy-based language. The Gradle scripts are managed alongside the product in the source code repository, so whenever a change in the source code requires a change in the build the software developers are able to make the necessary changes to the build configuration. The Gradle builds can also be executed locally on the developer's workstations.

4.1.7 Deployment platform

The deployment platform in Avaintec's case is a Docker software container. The software components of the products are deployed onto the containers which are compiled of a pre-built base container images containing the basic dependencies for the installation. In the current setup the containers are running on Linux virtual machines as their hosts.

Using containers was preferred due to the nature of containers being versatile regarding to the platform where they are ran. In essence the containers can be deployed to different operating systems and cloud platforms with little or no modifications in the future if there's a need to do so.

5 Devops tools in recent literature

In this section of the study the most recent literature on the subject of DevOps tools and continuous Delivery and Deployment is examined in a systematic literature review approach. The literature review was done by selecting the relevant articles found with a forward reference lookup from two recent literature reviews on the subject matter; "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices",[Shahin et al., 2017a] and "Problems, causes and solutions when adopting continuous delivery—A systematic literature review", [Laukkanen et al., 2017a]. The relevant themes found within the articles are introduced and processed in this chapter within the grouping done by the relevant research question.

5.1 Theme classification

Reading the selected papers resulted in a set of common themes around the research questions. Multiple common themes were discovered in the papers. The themes from the papers found significant for this study are presented in Table 3. The papers are identified in the table with identification tags S1-S11 that correspond to those presented in Table 1.

Table 3: Article theme matrix

Paper identification	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
RQ1: How do software development teams experience the usage of DevOps pipelines?											
Culture and organisation											
Organisation and process	x		x				x				x
Visibility to the whole company	x		x								
Culture and communication	x		x								
Development and operations joint venture	x		x				x				x
Technology											
Process and / or guidelines when adopting	x	x	x								
Learning curve							x	x			
Reliability											
Reliability of environments / installation					x			x			
Reliability of configuration management	x		x		x				x		
Human error reduction, quality improvement	x		x			x		x			
Quality improvement through reliability	x		x								
Time											
Less manual configuration			x		x			x	x		
More possibilities when automated	x		x			x					
Additional work initially required		x	x	x	x		x			x	
Cycle time reduction	x		x			x		x	x	x	
RQ2: How does a DevOps toolchain evolve during its lifespan?											
Pipeline evolving with the product			x	x			x				
Gradual building of pipeline							x				
Preservation of maintainability				x				x			
Difference of deployment environments	x		x								
Improvements in automated testing		x	x			x		x			
Adding more automation								x			
RQ3: How should DevOps pipelines be used for them to be helpful for the development team's work?											
Management decisions to allow automation		x					x				
Correct usage of tooling	x						x				
Product built for deployment								x		x	
Gradual building of pipeline			x				x				
Paper identification	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11

The themes are classified in groups under the research questions. However some themes do overlap significantly within the research questions, thus those themes are addressed in multiple classes.

5.2 Class 1: How do software development teams experience the usage of DevOps pipelines?

This theme is represented in four parts due to the vast amount of themes experienced by software developers. The themes are Culture and organisation, Technology, Reliability and Time. The mapping of findings to these themes is also represented in Table 3.

5.2.1 Cultural and organizational themes

These themes occur often in research papers about DevOps and development process automation.

[Zulfahmi Toh et al., 2019](S1) describe *organisation and process* related issues to be the most discussed ones and that the existing methods and processes usually aren't suitable for DevOps. [Chen, 2017](S3) points out that even though we know that the organisational structures can be a problem, there is no model or target for an optimal organisation structure that would work with DevOps pipelines. Commitment of the development teams is pointed out as the most relevant element of Continuous software engineering, a bundle of continuous integration and delivery presented by [Johanssen et al., 2018](S7). Flexibility and optimization of the organization to align with DevOps practices is pointed out by [Shahin et al., 2018](S11), as well as the importance of searching for the suitable skills to the company and arranging roles efficiently in development teams. An inflexible organization can become a blocker for DevOps implementation.

Visibility to the whole multi-functional team including the stakeholders and management is deemed important by [Zulfahmi Toh et al., 2019](S1), so that all parties are aware of the possible issues and statuses in the build automation pipeline. Thus these stakeholders will be able to react faster if needed. [Chen, 2017](S3) presents that the same theme of visibility also helps to drive interest of the stakeholders towards further developing the DevOps pipeline if they find the information presented by it useful.

Culture and communication within the company is an important aspect for DevOps to succeed. [Chen, 2017](S3) raises the issue of communication difficulty between the developers and the system engineers, as they speak a "different language" due to their background differences. Similarly, cultural differences and miscommunication can cause lack of trust within the organisation and operations may fail due to that, [Zulfahmi Toh et al., 2019](S1)

claim.

Papers S1, S3, S7 and S11 all mention the theme of *Developers and Operations having a joint venture* in the development process.

While [Zulfahmi Toh et al., 2019](S1) uses the wording of "forcing developers and operations team work together" in order to have the required sharing of information within the process, others see the case as important too but with less drastic wording. [Shahin et al., 2018](S11) Address the importance of software engineers having to be able to "expand their roles" from bare software design to be also able to work together with Operations people, taking part in the operations related designs too.

Keeping an open mind and and being able to adopt to changing, shared rule sets are raised as the pain points for developers when adopting continuous development strategies according to [Johanssen et al., 2018](S7). In the study it's pointed out that developers with longer working history may have more difficulty in adopting to a new way of working within a team that has people and tasks from outside the scope of pure development work.

[Chen, 2017](S3) Raises the point of building the development team with "multi-disciplinary" members from the ground up. According to the study, the deployment automation requires a great deal of knowledge in operating systems and application configuration which the developers don't necessarily possess. Chen points out that if a team does not have members with operations experience, the adoption to Continuous Delivery will take longer as the knowledge will need to be acquired by the development team. By adding a team member with experience in system administration and operations they saw accelerated progress and also witnessed relief of tension in the development team.

5.2.2 Technology

Technology holds an important role when adopting DevOps practices. As with any novel technology, the guidelines or best practices don't necessarily exist for everything and the learning curve can be steep.

The existence of *process and guidelines* for DevOps practices is mentioned by [Chen, 2017](S3) as he addresses the lack of them in the scope of tool selection and designing the stages for the pipeline. Chen mentions the necessity of additional research on that area to find best practices and to gain possibilities to evaluate tools from different vendors according to one's needs. On the organizational level, guidelines are produced when a deployment automation system is taken into use but according to [Zulfahmi Toh et al., 2019](S1), the guidelines are insufficient in half of the cases. Zulfahmi et.al. mention for example insufficient incident management process as a time consuming factor. They also point out that research is yet to standardize the practices in DevOps, and that the ambiguous definitions of DevOps can create confusion for an organisation when planning an objective. This is also emphasized by [Shahin et al., 2017b](S2) as they state that the vendors of the available tools should try to find some sort of standards to make the adaptation work easier for companies.

When adopting any new practices, there's always somewhat of a *learning curve* involved in the process. S7 and S8 address the learning curve from a couple viewpoints. [Johanssen et al., 2018](S7) observed that one of the main challenges in DevOps adaptation is the compliance of developers with the new rule sets and the ability to evolve skill-wise, as development work requires continuous learning already without the Ops aspect. As the tooling is often very flexible and provides possibilities for complex configurations, which is good from the point of what can be achieved, it also requires a lot of work to configure. These are pointed out by [Zhang et al., 2018](S8) as reasons for altering tooling or redesigning a pipeline, as one of their presented

"unmet needs" for Continuous Delivery and Deployment is that the pipeline should be "easy to learn and configure".

5.2.3 Reliability

Reliability is one of the main targets in deployment automation. Configurations, installations and environments can be more reliable when the human error factor is less present, comparing to manual deployments.

Reliability of environments and installations is mentioned in S5 and S8. Rollback mechanisms, automated scripts that are tested multiple times and standardized deployment processes make for a reliable production environment states [Zhang et al., 2018](S8). In their article about using Docker containers as the deployment platform, [Garg and Garg, 2019](S5) provision docker containers as a solution to automating the environment installation and removing the worry on configuration differences.

As the next step to an environment installation, the *Reliability of configuration management* is an important aspect of a DevOps pipeline. [Zulfahmi Toh et al., 2019](S1) state the possibility of versioning and full control of the configurations as an important asset in DevOps. Reducing configuration drifts is also pointed out by [Chen, 2017](S3), as the automated configuration works the same way every time.

Software development is work done by humans, and thus *human errors* are inevitable. Reduction of these errors can be achieved by the DevOps automation practices by automated testing and staging as proposed by [Zulfahmi Toh et al., 2019](S1). Automated testing is also found to be reducing manual errors in testing[Laukkanen et al., 2017b](S6).

Avoiding bugs in the source code and catching them early, before they affect other developers work is found by [Zhang et al., 2018](S8) to be one of the big benefits of Continuous Integration part in DevOps. Less bugs in a software product equals to *quality* in software which can be achieved through

DevOps practices according to [Chen, 2017](S3). In the study, Chen presents a massive reduction of 90 percent in open bugs in the software, compared to manual testing and deployment.

5.2.4 Time

Time is valued by companies and their employees. If tasks can be done faster, it allows fore more of other things to be done. Automating phases in software development and reducing the amount of manual work is one of the core themes in DevOps.

Reducing manual configuration has a positive effect on the time it takes for a software to be released into a test or production environment. The proposed Docker container solution by [Garg and Garg, 2019](S5) separates the need for manual configuration by design, as the containers can run on any environment without need for any special configuration.

In the study by [Luhana et al., 2018](S9) an automated delivery pipeline reduced the amount of time for example in creating screenshots for an app store by more than an hour when comparing manual work to an automated task. However also for more traditional deployment environments, [Chen, 2017](S3) and [Zhang et al., 2018](S8) find that automating the configuration for developers test environments gives 20 percent of time back to other tasks.

Getting free time back for other tasks allows for *more possibilities*. More tests and tasks can be implemented to the automated pipeline as stated by [Zulfahmi Toh et al., 2019](S1), things that could not otherwise be done manually in reasonable time. [Chen, 2017](s3) refers to this possibility too, as there is now more free time, then more things can be automated. However at this point it's good to keep in mind the point by [Zhang et al., 2018](S8) that the pipeline design can become too complex and thus hard to comprehend for example for team members that haven't been there to develop the pipeline

from the beginning.

The *initial requirement for additional work* when setting up a DevOps pipeline is recognized by multiple papers(S1-S4, S7, S10) of those studied. A lot of changes may be required to make for example the server infrastructure compatible with CD [Chen, 2017](S3). The large amount of tools and getting them to function properly together with the earlier stated lack of proper guidelines will consume a lot of resources [Shahin et al., 2017b](S2). The common theme is that the setup of a DevOps practice takes a lot of effort [Johanssen et al., 2018](S7), which can be difficult to sell to stakeholders since it's a significant cost factor [Chen, 2015](S3). On the other hand, having the proper resources may not completely eliminate the issues in CI adoption as stated by [Laukkanen et al., 2018](S10) in their comparison of a large company and a startup adopting to modern release engineering practices. Issues may occur even with high resources if the initial requirements for the DevOps implementation are set too high.

In addition to monetary investments, the organization changes required may also take their toll [Zulfahmi Toh et al., 2019](S1). Developers may easily get too much on their plate in an adoption process to DevOps. New concepts, methods, techniques, tools, processes and environments require time for adaptation [Steffens et al., 2018](S4). If rushed too fast, these may lead to the cultural and organizational issues mentioned earlier.

As a proposal to these adoption issues, [Chen, 2017](S3) used an incremental way of introducing DevOps practices. To begin with, they started the adaptation to DevOps with one project type, which was significant enough so that the improvements to it's development cycle would attract attention in the developers and also in the management. Then, as the idea of DevOps had been "sold", they were able to proceed incrementally to other project types.

As time is of essence in software development today, it's optimal to

have a short *cycle time* from development to production. According to the studies, DevOps and CDE/CD practices make it possible to release faster [Zhang et al., 2018](S8). Rapid customer feedback to frequent releases can make the work of the development team easier as it's possible to tackle the possible issues early on as stated by [Zulfahmi Toh et al., 2019](S1) and [Chen, 2015](S3).

As faster releases allow for rapid customer feedback, it was found by [Laukkanen et al., 2018](S10) that faster release times lead to an increase in defect reports. They state that the increased defect reports don't necessarily mean that the quality has suffered in faster releases, but the issues may be such that they can only be detected when already in production. So it is possible that even with more rigorous release testing these issues might still have gone through to production.

Feature freeze periods are a commonly used way of keeping the software stable when preparing for releasing. [Laukkanen et al., 2017b](S6) discovered with their case organization that using test automation in a deployment pipeline reduced the feature freeze time by 56 percent. This was due to continuous automated end-to-end testing in the development phase, and thus fewer changes were needed during the feature freeze period to make the software releasable.

Some production deployments require additional work due to requirements of a distribution platform. In the deployment workflow presented by [Luhana et al., 2018](S9), a total saving of one hour and 43 minutes per release was gained in deployments of a mobile application as the developers, for example, no longer needed to take the screenshots of the application manually for the application store platform.

5.3 Class 2: How does a DevOps tool chain evolve during its lifespan?

A DevOps tool chain should be in a state of *continuous evolution with the software product* itself. As products get new features or new technologies introduced to them, the tool chain and the CI/CD pipeline should be developed from the beginning [Steffens et al., 2018](S4) to fit those needs. This should allow for addition of features without major time or cost factors[Johanssen et al., 2018](S7).

One of the challenges here is to keep the pipeline evolving. Development teams can often lose momentum in the CDE/CD pipeline development as time passes. [Chen, 2017](S3) proposes a technique of a "Visual pipeline skeleton", in essence drawing a schematic of a perfectly automated pipeline at the beginning of the CDE adaptation project. Some parts of the planned pipeline will be implemented immediately, while some remain unimplemented. With this visual representation of what could be done the interest to continue developing the pipeline can be better kept up. With this sort of *gradual build of the pipeline* the evolution of the pipeline should stay as an ongoing project, and the possibility to develop towards a Continuous Deployment remains.

One of the aspects that is important in a tool chain is the *preservation of maintainability*. [Steffens et al., 2018](S4) give an example of presenting a new test case to the pipeline, it needs to be possible with reasonable effort. Also new non-functional requirements, for example new legislation may occur, and the pipeline needs to be able to adapt to those needs. If the maintainability of the pipeline cannot be preserved, it may become impossible for it to evolve. In this case a pipeline that is too difficult to maintain should be changed to a workflow that's easier to maintain and update [Zhang et al., 2018](S8).

The *differences in deployment environments* can purely come from the purpose that the environments are originally meant for. A test or CI envi-

ronment meant for real-time testing during development is naturally lighter than a full production environment. Starting to build a pipeline from these simple environments and then evolving it to support the complex production deployments is a good practice, and a pipeline should be able to provision all types of environments quite effortlessly [Chen, 2017](S3).

Issues may arise during the evolution of the pipeline as the production environments may grow in complexity and the test environments can stay quite similar. For example test cases may behave differently on a simple environment compared to a production environment [Zulfahmi Toh et al., 2019](S1). To avoid these issues, using a staging environment as similar to the actual production environment in the release phase is recommended.

Automated testing is a core aspect of build and deployment automation. Better support for automated testing in the pipelines was deemed necessary in two of the studied papers. [Zhang et al., 2018](S8) found in their study that additional automated testing would be needed for the build configurations, as the only way of testing the build was to run it and it took a long time. The pipeline evolution should allow for such tests to be added. Additional testing needs for the evolving and different environments were found by [Shahin et al., 2017b](S2), as the automated tests in the pipeline had not evolved with the environments and thus errors leaked through. In the study it was also found that a need for evolution was in the possibility of parallelizing the testing tasks.

Adding more automated testing as the pipeline evolves is a natural course of action. [Chen, 2017](S3) proposes in an example case of a gradually built pipeline that the missing tests are visualized as "gaps" in the deployment pipeline, and they thus motivate the developers to fill those gaps with the missing test tasks. This way the benefits from the pipelines existing tasks can already be experienced and as the pipeline evolves the additional benefits will come on top of that.

A concrete example of such evolution with the testing is given by [Laukkanen et al., 2017b](S6), in their study adding test automation between minor versions first to 40 percent and then to 75 percent decreased the feature freeze time in the release stage by 24 days on average. Naturally this is due to other automation tasks too, but testing played a major part.

What more can be automated? Unless some tasks in the automation pipeline require management acceptance [Shahin et al., 2017b](S2) or if the deployment environments are not designed to be accessible by automation, for example some customer environments [Shahin et al., 2017b](S2), all phases of a software product deployment should be possible to automate, in essence do CD. If a workflow does not allow for a full automation but requires manual steps (CD), it should be changed and something new tried instead [Zhang et al., 2018](8). A good example of the possibility of automating everything is in the study by [Luhana et al., 2018](S9), where taking screenshots of the application for a mobile app store listing was automated.

5.4 Class 3: How should DevOps pipelines be used for them to be helpful for the development team's work?

Initially implementing DevOps and the start of using automated deployment pipelines is a *management level decision*. The quality concerns and delivery processes that keep the management of a software company insisting on manual phases in the deployment need changing [Chen, 2017](S3).

As the quality control is implemented into the automated testing part of a DevOps pipeline carefully, the process should become trustworthy and allow for the quality concerns to be lifted. One of the main causes for not implementing a full deployment pipeline for a product is that production deployment is considered a business decision, or it can have a highly bureaucratic process from the company or the customer side related to it [Shahin et al., 2017b](S2). As manual work is lesser and the quality control

can be implemented by automation, this sort of approach can also be compelling to management teams and stakeholders [Shahin et al., 2017b](S2). To achieve this, however, the changes in processes are required.

For the implementation of the pipeline, as seen in the earlier chapters, resourcing is needed. For a DevOps implementation to be successful the company management needs to assign the proper resources to the project [Johanssen et al., 2018](S7). The "expert drop" method introduced by [Chen, 2017](S3) is a way of coping with the resourcing in the initial phases of building a deployment pipeline. It's important to establish a team with sufficient skills for the work, and to not assume that developers with no earlier experience of DevOps and build automation would be able to build an efficient deployment pipeline in a timely fashion.

Tooling is key in DevOps pipelines. It's possible to choose a bunch of tools, but you need to *use them correctly*. First of all, the tooling resources need to be managed. As existing guidelines are vague, a strategy and a guideline in the tool usage needs to be established when designing the pipeline [Zulfahmi Toh et al., 2019](S1). Using a combination of tools from the same vendor may at times be a way to easier success, as they often have synergy benefits [Johanssen et al., 2018](S7).

Controlling the hardware resources used by the tooling is also essential. A large amount of resource overhead may accumulate if a software is being built too often [Zulfahmi Toh et al., 2019](S1). This can consume resources from operations of other products running builds on the same platform, or just plainly invoke unnecessary costs from hardware usage.

Sometimes companies struggle with tools they have been using for a long time, but that are no longer compatible with the new tool sets or workflows. This is a common case in implementing DevOps. Tools itself are not the issue here, but it's the integration. For example an issue tracking system may have worked for internal incident management, but not for the development

purposes [Johanssen et al., 2018](S7). Also the same issue tracking system may communicate nicely with some other system that the company has been using for years, but it may not cooperate proficiently with the new CDE/CD automation server wanting to link build results to the related issues.

Apart from designing the pipeline, the software product should also be designed to be suitable for deployment pipelines. Monolithic software designs are a challenge for the success of CDE/CD, however it does not make it impossible [Shahin et al., 2018](S11). This requires for the development team to have customized tool sets, high test quality and backwards compatibility in feature additions.

Generally when designing software for CDE/CD the design decisions from early-on development phases should be done together with the operations personnel and the operational concerns should be prioritized. It's mandatory that the development of the software starts with the understanding of the environments, regulations and how the software is to be deployed in general [Shahin et al., 2018](S11). This calls for the importance of motivating the Developers and Operations personnel to have interest in each others working domains [Zulfahmi Toh et al., 2019](S1).

Designing and building a software from the beginning by keeping the deployment requirements in mind is clever in the sense that if int he future the software needs to be deployed using a CDE/CD pipeline, it can be done. For legacy software the transition is difficult, for example as stated earlier about monolithic designs. The adaptation to DevOps can happen in a *gradual* way and it should, since migrating all of a company's legacy software at once is usually just impossible or would take years [Chen, 2017](S3). On-demand adaption is proposed as a course-of-action by [Johanssen et al., 2018](7), their message being that the additions to the deployment pipeline should be useful for the development team's work at that point in time and the decisions of other additions should be postponed to the time they are necessary.

6 DevOps pipelines at Avaintec

For this section five semi-structured interviews of people working in development teams were conducted to get an overview of the implementation, usage and evolution experiences of the DevOps tools and the build and deployment pipelines at Avaintec. The transcribed theme interviews were coded and the common themes found for the coding are presented in this chapter. The initial codes and their focused correspondents can be seen in table 4.

Table 4: Initial and focused coding of interviews

Focused coding classes	**Initial coding classes from the interviews**
Reliability	Trust for the environment
	Reliability of the deployment
	Healthy state at all times
Feedback	Feedback from the builds
Knowledge / documentation	Lack of Documentation
	Lack of guideline
	Correct/incorrect usage of tooling
	Pipeline Design
Test automation	Need for more automation
	Need for more automated tests
Learning curve / unfamiliarity	Understanding
	DevOps Joint operations od Dev and Ops
	Difficult to comprehend the whole system
	Difficult if not continuously involved with the pipelines
Manual work	Less hard-coded
	Less manual work
	Optimal when automated
	Optimal when least work
Technological resources	Not suitable for standalone setups
	Environment similarity should be automated
	Need for having enough environments
Product design for DevOps	Slowness in building
	Product design not for deployment

In the following chapters the themes in the focused codes are addressed. The different interviews are referred to with the interview identifier (I1 - I5), according to which they are also presented in the code matrix in table 5

Table 5: Interview theme matrix

Theme / Interview ID	**I1**	**I2**	**I3**	**I4**	**I5**
Reliability	x		x	x	x
Knowledge / documentation				x	x
Feedback			x	x	x
Test automation	x			x	x
Learning curve / unfamiliarity	x	x	x		x
Manual work	x	x	x	x	x
Technological resources	x	x	x	x	x
Product design for DevOps			x	x	x

6.1 Reliability

The theme of reliability came up in most of the interviews (4 of 5). The installations of the environments and the configurations of the installations were considered trustworthy when done through automation. When the deployment has been done by the automated pipeline, it can be trusted: "I don't have to waste my time installing and checking things if they are how they are supposed to be, I know that this is the proper thing I'm testing."(I1) This reflects well to the literature as environment and configuration differences are found to be well eliminated by using an automated deployment.

Merging the changes with other developer's work is considered reliable and the test environments are seen as good places to verify that the software will actually work also in production, as that is not possible, especially with some products, on the developer's own development environment.

When comparing to previous approaches, the automated pipeline to the test systems is considered a blessing as the earlier way of working was a collection of scripts that did the installation of a certain version of a software. As the scripts were difficult to operate, developers had a tendency to not use them thus not deploying all of the changes on to test environments. The

result of this was software in broken state and an unreliable development version. Now, as the code is automatically integrated, built and installed onto a test server after every commit to the development branch, problems can be caught immediately. This early catching of bugs with the help of continuous integration is also in line with the findings from recent literature in 5.2.3, where it was also found that a great amount of them can be caught with the help of automation.

6.2 Knowledge / documentation

In recent literature the lack of guidelines and ready-made processes is presented as one of the key adoption issues with DevOps 5.2.2. In the interviews this theme was also emerging. One of the interviewees found that the documentation for the tools selected for the build automation have not been easily available or they have often been outdated or just not working in practice.

The lack of existing guidelines seems to have emerged in Avaintec's case in a way that some of the tools are used incorrectly: "I don't think that it's used the way it's meant to be used."(I5) For example in one product the configuration management of different environments is done in one separate repository with branches for different environments, thus not allowing for real version control. Also the environments have separate build configuration files, as they could be just stored in the source code repository in the original configuration files. This approach could help removing unnecessary overhead.

Originally, both the build and deployment tasks in Avaintec's Deployment pipeline were all done in the same build plan in Atlassian Bamboo. This was a fast setup done with the knowledge that was available at the time the pipeline build was started. Over time there has been improvements in multiple sectors of the DevOps pipeline. The build and deployment phases have been split to separate phases to allow for more automated builds, and to allow for selection of the deployment environment for each build. One

interviewee commented on this evolution as "It's smoother, now we have the builds there clearly which then go to the deployments. And it's also possible to do rollbacks now."(I4)

Also the building of the docker containers has been split into building a base-image first, which contains all of the dependencies and the requirements for the software installation. Only after that the software itself is installed to the container using the base-image as the starting point. This saves time in the container build as overhead is reduced.

In the interviews these changes were found to make building and deploying eventually easier. In retrospective, an "expert drop" method introduced in the current literature (5.4) could have possibly reduced these mistakes done in the Devops Pipeline development. However as the first implementations may have not been implemented in the best possible way, they started to give immediate value to the projects, which is important for keeping up the momentum in the pipeline development as proposed by [Chen, 2017] earlier in 5.2.4.

6.3 Feedback

In three out of the five interviews, the feedback that the pipeline gives to the developer was discussed. As most of the deployments are triggered by automated deployments, the monitoring and feedback purpose from the pipeline is one of the most essential use cases. When asked about usage of Bamboo, (I3) responded with the following use cases: "..Checking what has been deployed, what deployment has broken the code and who has pushed the code". Getting this immediate feedback from the build system is important. As the evolved version of the pipeline builds all branches of the source code automatically, the problems can be found fast: "It's nice that the builds are ran automatically so that if the build fails, you know you can blame yourself."(I4) "It's quite neat that every branch is built automatically."(I5)

"You can immediately see if you have made some mistake that breaks the build."(I5)

In the reviewed literature, it was found that the automated environments do reduce the amount of bugs that pass through the development phase, in some cases by even 90 percent. In Avaintec's case the feedback to the developer from his/her code does exactly that. The percentage may not be that high but certainly a lot of problems are detected early on in the development phase thanks to the automated builds.

6.4 Test automation

To get the aforementioned feedback from the system, test automation is important. The amount of automated tests varies between different products, some have more automation while others have more manual test cases. For the automated test cases, the unit tests are integrated to the build phase of the deployment pipeline. The interviewees found it to be useful so that the developer can get immediate feedback from the code.

On the other hand, interviewees see the need for more automated testing with all products. There are not enough unit tests in the pipeline, and also API testing is troublesome: (I4)"Testing the API:s through UI:s does not make much sense" and also testing the basic functionalities should be automated: "An automated set of some basic functionality checks (would be nice to have at some point)"(I1)

The amount of automated testing is very rarely satisfactory in automated deployments. In the reviewed literature (5.3) better support for automated testing is set as a requirement, and also the adding of tests gradually to the pipeline is found to be a good approach to the issue. At Avaintec the development teams are well aware of the lack of automated testing in the products. The gradual adding of tests happens when it's found to be so important for the product development that it has to be done, as in the

on-demand-adaption proposed by [Johanssen et al., 2018] in 5.4

6.5 Learning curve / unfamiliarity

In the reviewed literature, the learning curve in the adoption to the new tools and working methods of DevOps was presented in two themes. First, the adaptation of people to work in the different contexts, in essence developers contributing to Operations tasks and vice versa. Second the complex configurations of the build and deployment automation systems were addressed as a possible issue. In the interviews conducted at Avaintec both of these aspects were found.

For the context change, an interviewee commented the following: "In the beginning it can be a bit scary like what's going on under the hood as things are going fast. But then it's quite fascinating.."(I6) So the change of context can be intimidating at first but when people get on board, the benefits are clear.

In the interviews it was obvious that some of the configurations in the system are hard to find if there is a need for changing something, likely due to the falsely produced configuration repository model that was addressed earlier. Also a point of counterintuitivity related to an update adding features to the pipeline was raised: "when we updated a year ago it became super counterintuitive, so a year ago we had an old version and it was more straightforward."(I5) However, the interviewee continued that ".. then I got used to the new one and it allows many more things in terms of continuous integration". With the updated automation pipeline, some interviewees also find that the systems have become easier to use and somewhat faster.

These issues were maybe partially explained by a few comments from the interviews: "the difficulty comes from not working with the tools at all times" "Of course someone else configured it for me". In these parts it's clear that engaging the whole team to the development and the evolution of the

pipeline, in essence doing DevOps, has somewhat failed. For some parts the pipelines have become tool sets for the development team that someone else put up that they don't know enough about. This is one of the core issues that has been emerging throughout this study, also in the way that multiple interviewees don't remember the changes that have been introduced to the pipeline. Naturally this tells that the involvement of the whole development team has not been achieved.

6.6 Manual work

Reduction in manual work is the most obvious reason to do any automation. The build automation in Avaintec has reduced manual work in multiple ways according to the interviews. Interviewees are happy that a lot of the time-consuming, repetitive installation tasks that were done manually in earlier projects can now be done automatically with just a click of a button: "Basically you can do a lot of stuff with one click"(I1). The time reduction experiences developers have at Avaintec seem to have same characteristics with the ones from the literature in 5.2.4. The possibility to get more done in the same time seems to overtake the amount of time used for the initial setup of the pipeline.

The possibility to deploy the build result from each branch to a production-like test environment is found to be "a nice touch" (I5). As the manual phases are reduced, the aforementioned reliability aspect also links in to this. Less manual work means less mistakes and thus less broken states in source code. This has a direct relation to the Reliability aspect discussed earlier.

As the build and deployment tasks are automated, the feedback presented earlier comes every time without a need for any manual steps after pushing your code to Bitbucket to the branch you are working in. The fact that the developer's code will end up where it's needed when it's needed was found to be a great reduction in manual work. Also the effortless way the automation

gives the feedback is mentioned: "It's good that there is an automation that does not require too much follow-up."(I3)

As found in the researched literature 5.3, the deployment pipeline and the DevOps working methods are in a state of continuous evolution and they should be. At Avaintec manual work is still required in multiple phases of the deployments. Already earlier the lack of sufficient automated testing was mentioned. Also the processing of dependencies in the applications, mostly node modules, is still done manually, which creates a lot of work. As the evolution is ongoing, the newest additions to the product portfolio are actually using an automated way of maintaining the node modules.

An another evolution to the deployments were new Gradle scripts that were made to make the deployments to personal development servers easier in Signhero. This is a good example of what was found in literature too, if a tool or a way of working does not work for the purpose, it should be changed.

6.7 Technological resources

As in the reviewed literature 5.3, the interviewees also found that it's good to have enough environments for different use cases, thus allowing development and testing in different branches without interfering with the others: "They can test the latest version in development branch, as well as the environments for release-upcoming and release-current where they can verify upcoming releases without taking into account possible interference from the development branch"(I5). One interviewee points out that it would be good to have more of close-to-production environments as issues getting into the development branch can form a bottleneck in the development.

From the testing side, as a lot of manual testing is done before merging a software release to the master branch, the environments need to support that. Sometimes manual testing may take a long time, and as unoptimal

that is, it's necessary. During this time the product development needs to be able to move on and thus it's good that there's a separate server for testing between releases. This allows for the feature freeze periods to be shorter, as seen in the studied literature (5.2.4) too.

The personal development servers were also found practically useless by one interviewee, as the same thing could be done on your workstation as on the personal development server. Another interviewee found these environments to be unstable as the installations aren't automated an they can thus differ significantly from the production or even the automatically deploying development environments. This issue was partly mitigated in the Signhero development with the Gradle scripts designed for a unified way of deploying to the environments. However a third interviewee thought that these environments should not be automated completely, since at least for now the automation has been too slow for testing minor changes in User Interfaces, for example.

6.8 Product design for DevOps

Monolithic designs are a challenge for DevOps and especially for continuous builds and deployments. In the reviewed literature (5.4), [Shahin et al., 2018] however propose that it's not impossible. With highly customized tooling and pipelines it surely is possible, as has been proven with Signhero and Datachief at Avaintec. The monolithic core however does have it's issues. In the interviews, long build times were pointed as an issue as the monolithic core has to be completely rebuilt to a new installation package when changing some parts of it. It was also pointed out that "It's not very common that you need to deploy a solution to a server to be able to test it"(I5).

These issues often occur when the product is not designed from the start to meet the needs of CDE/CD and DevOps. In literature (5.4) the proposal is to design the software from the very beginning to support these

needs, in essence build software for DevOps. At Avaintec this has been taken into account when the design of the new cloud platform was started, thus learning from the past and evolving the pipeline. The new platform allows for different parts of the software to be run, built and developed independently, thus making them microservices. As found in the literature, it can also be discovered from the interviews that it's very hard doing DevOps with monoliths and microservices seem to be a part of a solution to allow it.

7 Discussion

In this section possible development possibilities and solutions for better DevOps practices are discussed, and the threats to the validity of this study are presented and analyzed.

7.1 Development possibilities

An interesting question that came up is how to motivate development teams for the work within the DevOps context and how to make possible for the development of the deployment pipeline alongside with the development of the product, without causing too much interference to the product development itself. Naturally this can be solved with adding resources to the product, in essence the way that it was done in the reviewed literature and in the studied case, but would it be possible to facilitate this without much additional workforce? Maybe if the documentation and know-how about the building of the pipelines and standards were more thorough in the industry.

Another possibility for future studies is the question on how to motivate customers to move their production environments to be automatically deployable, in essence to continuous delivery. In the literature it was pointed out that many times this might not be possible due to restrictions in environments or legal aspects, or just the need of manual verification before each

release. Also the aspect of not wanting to have updates all the time came up in the literature.

One thing to note about the product development from the experience in the case study and from the literature is that Monoliths and DevOps practices with automated deployment pipelines don't blend too well. So if possible, monolithic products should be avoided in this context since they create a lot of additional work in the pipeline development as a lot of customisation is required.

7.2 Threats to validity

In this section the threats to the validity of this study are discussed and evaluated using the classification of validity threats by [Runeson and Höst, 2008].

Construct validity depends on the concepts of the study being understood the same way by the researcher and the Interviewees. In this study a member checking round was performed with the findings by running the findings through all of the interview participants. No major adjustments or corrections to the findings were needed after the interviewees had studied the results. However it is possible that the constructs defined by the researcher have had an impact on the viewpoints of the interviewees.

Internal Validity examines the causal relations of the findings in the study. As the case study was conducted in a setup where a lot of new tools and working methods were introduced, the experiences of a team member may have been affected by some tooling or some method more than what the same tool has affected some other team member. In essence similar experiences may have been caused by different things. On the other hand, in the scope of this study the validity can be considered good for analyzing the tool chain in general as long as the causes are within the scope of the DevOps tools and working methods.

External validity defines if the results from a study can be generalized to

other settings. In this study the papers researched and the people interviewed were of a somewhat narrow scope, however the recurring themes are likely to occur in other setups too as they seem to be recurring themes also in earlier studies. For example the systematic literature reviews that were used for the selection of papers for the literature review contained similar themes.

Reliability as a validity aspect studies if the results of the study are dependant on the researcher(s) conducting the study. As the researcher, my role as a system administrator responsible for the development of the DevOps pipelines will pose a risk to the reliability of the study. Thus the interview questions as well as the researcher as the responsible person of the DevOps pipeline implementation conducting the interviews may have had impact on the results. The goal was to keep the research neutral and the interview participants were assured that they are very much encouraged to speak their minds, which, when looking at the output from the interviews, they did.

8 Conclusions

In this chapter the conclusions from this study are presented. These conclusions are grouped under the themes of the research questions to give a clear view on what were the core findings in this study for the research questions. In general it should be said that the themes extracted from the reviewed literature were quite similar to the themes that came up in the interviews, thus making the case study well comparable with the recent literature.

How do software development teams experience the usage of DevOps pipelines?

In both, the researched literature and the conducted interviews, software development teams find the DevOps methodologies and the tools related to that to be useful and that the tool sets enable new possibilities through the automation. Reducing repetitive and manual tasks is seen as an important

advantage, as well as the reduced possibility of errors due to less manual interaction.

However in the reviewed literature and during the interviews it was clear, that the learning curve for the new methodologies and for the pipelines and tools used for the Continuous Integration, Deployment (and Delivery) is steep, and can cause issues. To mitigate these challenges, the literature proposed taking the tools into use in small increments and to keep them in constant evolution. Also the importance of having all team members developing the pipelines together was emphasized. At Avaintec the development of the pipelines has been more or less done in this way, although it was clear that the developers weren't familiar enough with the existing tools and methods in use. Also at some stages when the build and deployment pipeline has changed during its history, developers have experienced that the changes were difficult at first, but after a while they were useful and allowed for more possibilities.

The amount of initial work was something that came up from the literature and also from the interviews, as some commented that they are aware of the work done in developing the automation for them. As for the learning curve, for the initial work literature also proposes that the incremental approach on developing the pipelines should be used.

How does a DevOps tool chain evolve during its lifespan?

A DevOps tool chain is in a constant state of evolution during it's lifetime. The study indicates that there are multiple places for evolution, for example automated tests, build and installation methods, branching models and so on. All of them are progressing with the changes in the product, the requirements from the environment or even with external changes related to legislation.

An important aspect for the evolution is that it should always be possible and the pipeline should remain in a maintainable state. If some requirement cannot be relatively easily fulfilled with the pipeline, then the corresponding

tool should be replaced. It was indicated in the reviewed literature that it's a good practice to develop the tool chain and pipeline in small increments to get immediate results without too much initial work. In the case at Avaintec the amount of initial work was significant, but the incremental evolution after that has been proving much improvement in the places there has been need for improvements.

In the interviews it was indicated that many interviewees had not noticed many changes during the evolution of the pipeline. On one hand this can be considered a good thing, since things have apparently worked smoothly after the changes if the changes have not been noticed. On the other hand it tells that the DevOps way of working has not been conducted in the pipeline development as the whole development team is not aware of the changes. Thus the study indicates that there is a possibility that the tool chain could be better with more interaction between the Developers and Operations people. The ways to do this, from the literature, are to motivate the developers interest in the tool chain and to provide additional time from the projects to the pipeline development.

How should DevOps pipelines be used for them to be helpful for the development team's work?

As noted in the previous section, a DevOps pipeline or tool chain should always be a joint development operation between the Developers and Operations personnel. As the Developers work with the same pipeline daily, they have a lot of input onto what could be done differently and thus make the pipeline even better to support their needs in their daily work.

The product should be engineered from the beginning to fit for build and deployment with an automated set of tools. Microservices are preferred over monoliths in this context since their deployment has less constraints as parts of the product can be built and deployed separately. This way the project team can work in a unified way with the product and adding new features

to the product and the pipeline should be relatively easy.

Adding features and automation should be carefully considered, everything is not necessarily worth automating. In most cases though, automation reduces the risk of human errors and thus it makes the environments reliable for the development team. In essence, the amount of automatically deploying environments should be sufficient to support all of the needs of the developers, but they should not cause resource overhead.

Also management level decisions are needed for allowing the resources to be used and also to allow for the developers to take responsibilities in the operations field. If in addition to Continuous integration and Continuous Delivery the company wants to do Continuous Deployment, the management is required to have trust in the pipeline and allow for automatic product updates without manual interference.

References

Balalaie et al., 2016 Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

Charmaz, 2006 Charmaz, K. (2006). *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Introducing Qualitative Methods series. SAGE Publications.

Chen, 2015 Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54.

Chen, 2017 Chen, L. (2017). Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software*, 128:72 – 86.

Ebert et al., 2016 Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *IEEE Software*, 33(3):94–100.

Garg and Garg, 2019 Garg, S. and Garg, S. (2019). Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 467–470.

Humble and Farley, 2010 Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st edition.

Johanssen et al., 2018 Johanssen, J., Kleebaum, A., Paech, B., and Bruegge, B. (2018). Practitioners' eye on continuous software engineering: An interview study. In *Proceedings of the 2018 International Conference on Software and System Process*, ICSSP '18, page 41–50, New York, NY, USA. Association for Computing Machinery.

Laukkanen et al., 2017a Laukkanen, E., Itkonen, J., and Lassenius, C. (2017a). Problems, causes and solutions when adopting continuous delivery—a systematic literature review. *Information and Software Technology*, 82:55 – 79.

Laukkanen et al., 2018 Laukkanen, E., Paasivaara, M., Itkonen, J., and Lassenius, C. (2018). Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering*, 23(6):3535–3577.

Laukkanen et al., 2017b Laukkanen, E., Paasivaara, M., Itkonen, J., Lassenius, C., and Arvonen, T. (2017b). Towards continuous delivery by reducing the feature freeze period: A case study. In *2017 IEEE/ACM 39th International Conference on Software Engineering*, pages 23–32. IEEE Press.

Luhana et al., 2018 Luhana, K. K., Schindler, C., and Slany, W. (2018). Streamlining mobile app deployment with jenkins and fastlane in the case of catrobat's pocket code. In *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pages 1–6.

Runeson and Höst, 2008 Runeson, P. and Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131.

Shahin et al., 2017a Shahin, M., Ali Babar, M., and Zhu, L. (2017a). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Shahin et al., 2017b Shahin, M., Babar, M., Zahedi, M., and Zhu, L. (2017b). Beyond continuous delivery: An empirical investigation of continuous deployment challenges. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '17, page 111–120. IEEE Press.

Shahin et al., 2018 Shahin, M., Zahedi, M., Babar, M., and Zhu, L. (2018). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering*.

Steffens et al., 2018 Steffens, A., Lichter, H., and Döring, J. S. (2018). Designing a next-generation continuous software delivery system: Concepts and architecture. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 1–7.

Webster and Watson, 2002 Webster, J. and Watson, R. T. (2002). Analyzing the past to prepare for the future: Writing a literature review. *MIS Q.*, 26(2):xiii–xxiii.

Zhang et al., 2018 Zhang, Y., Vasilescu, B., Wang, H., and Filkov, V. (2018). One size does not fit all: An empirical study of containerized continuous deployment workflows. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 295–306, New York, NY, USA. Association for Computing Machinery.

Zulfahmi Toh et al., 2019 Zulfahmi Toh, M., Sahibuddin, S., and Mahrin, M. N. (2019). Adoption issues in devops from the perspective of continuous delivery pipeline. In *ACM International Conference Proceeding Series*, volume Part F147956, pages 173–177.

A Interview protocol

- **Theme 1 Basic information and role**
  - Role in development team
  - Products you are / have been working with
  - Experience years on the industry
- **Theme 2 The DevOps Toolchain**
  - Describe the tools that you use in your work for development, deployment and operations, what are their purposes and how do you use them?
  - What are your experiences on using tool / tools xxx (selected tool(s) from the ones mentioned before)?

    follow-up if necessary; How do you feel about using it / them?
  - Has the DevOps toolchain changed or evolved during the time you have been using it?

    How have the possible changes affected your work?
- **Theme 3 Development cycle**
  - Describe your usage of the DevOps tools in the process of adding a new feature into the product through a real-life example.

    Is this a typical process when adding new features?
  - Describe your usage of the DevOps tools in a bugfix process through a real-life example

    Is this a typical process when fixing bugs?
  - When do you think a DevOps pipeline is useful for the development cycle?
- **Wrap-up**

– Is there something you'd like to add or mention around the theme of DevOps?

– Is it possible to contact you if there's a need for some clarifications?