An Assessment of DevOps Maturity in a Software Project

Master's Thesis
University of Turku
Department of Future Technologies
Computer Science
2017
Tatu Seppä-Lassila

Supervisors:
Antero Järvi
Sami Hyrynsalmi

UNIVERSITY OF TURKU
Department of Future Technologies

TATU SEPPÄ-LASSILA: An Assessment of DevOps Maturity in a Software Project

Master's Thesis, 80 p.
Computer Science
October 2017

DevOps is a software development method, which aims at decreasing conflict between software developers and system operators. Conflicts can occur because the developers' goal is to release the new features of the software to production, whereas the operators' goal is to keep the software as stable and available as possible. In traditional software development models, the typical amount of time between deployments can be long and the changes in software can become rather complex and big in size.

The DevOps approach seeks to solve this contradiction by bringing software developers and system operators together from the very beginning of a development project. In the DevOps model, changes deployed to production are small and frequent. Automated deployments decrease human errors that sometimes occur in manual deployments. Testing is at least partly automated and tests are run after each individual software change.

However, technical means are only one part of the DevOps approach. The model also emphasizes changes in organizational culture, which are ideally based on openness, continuous learning, and experimentation. Employees possess the freedom of decision-making while carrying the responsibility that follows. In addition to individual or team-based goals, each employee is encouraged to pursue the common goals.

The aim of this thesis is two-fold. Firstly, the goal is to understand and define the DevOps model through a literature review. Secondly, the thesis analyzes the factors that contribute to the successful adoption of DevOps in an organization, including those with the possibility of slowing down or hindering the process.

A qualitative case study was carried out on a system development project in a large Finnish technology company. The data consists of semi-structured open-ended interviews with key personnel, and the findings are analyzed and compared to factors introduced in previous DevOps literature, including the DevOps maturity model. The case project is also assessed in terms of its DevOps maturity. Finally, impediments and problems regarding DevOps adoption are discussed.

Based on the case study, major challenges in the project include the large size and complexity of the project, problems in project management, occasional communication problems between the vendor and the client, poor overall quality of the software, and defects in the software development process of the vendor. Despite the challenges, the company demonstrated progress in some aspects, such as partly automating the deployment process, creating basic monitoring for the software, and negotiating development and testing guidelines with the vendor.

Keywords: DevOps, Agile, continuous integration, organization culture

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

TATU SEPPÄ-LASSILA: Ohjelmistoprojektin DevOps-kypsyyden arvioiminen

Pro gradu -tutkielma, 80 s.
Tietojenkäsittelytiede
Lokakuu 2017

DevOps on ohjelmistokehityksen toimintamalli, joka pyrkii vähintämään ristiriitaa ohjelmistokehittäjien ja järjestelmäylläpitäjien välillä. Ristiriitaa luo kehittäjien tavoite saada julkaistua uudet ominaisuudet mahdollisimman nopeasti, kun taas järjestelmäylläpitäjien tavoite on pitää ohjelmisto mahdollisimman vakaana ja saavutettavana. Perinteisessä ohjelmistokehitysmallissa tuotantoonvientejä tehdään varsin harvoin ja tällöin muutokset ohjelmistoon ovat usein kasvaneet monimutkaisiksi ja kooltaan isoiksi.

DevOps pyrkii ratkaisemaan tämän ristiriidan tuomalla ohjelmistokehittäjät ja järjestelmäylläpitäjät yhteen jo kehityksen alkuvaiheessa. Tuotantoon viedään pieniä muutoksia usein. Automaatio vähentää manuaalisessa tuotantoonviennissä helposti syntyviä virheitä. Myös osa testauksesta on automatisoitu ja testit ajetaan kaikkien muutoksien yhteydessä.

Tekniset keinot ovat kuitenkin vain osa DevOps-mallia. Niitä enemmän painotetaan organisaatiokulttuurin muutosta. Ideaali organisaatiokulttuuri perustuu avoimuuteen, jatkuvaan oppimiseen ja kokeiluun. Työntekijöillä on vapaus tehdä päätöksiä, mutta myös vastuu niiden seurauksista. Jokainen työntekijä pyrkii työllään edistämään organisaation yhteisiä tavoitteita.

Työ pyrkii aluksi kirjallisuuden avulla määrittelemään mitä DevOps on, mitä osa-alueita siihen liittyy ja tarkastelemaan sitä eri näkökulmista. Tämän jälkeen eritellään mitkä eri tekijät mahdollistavat DevOpsin onnistuneen käyttöönoton organisaatiossa. Vastapainoksi eritellään tekijöitä, jotka saattavat hidastaa tai haitata DevOpsin käyttöönottoa.

Kirjallisuudesta löydettyjä tekijöitä verrataan laadullisessa tutkimuksessa tehtyihin havaintoihin. Empiirinen tutkimus suoritettiin suuren suomalaisen teknologiayrityksen tietojärjestelmäkehitysprojektissa. Tutkimusmetodina käytettiin projektin avainhenkilöstön haastatteluja. Havaintoja verrattiin myös kirjallisuudessa esitettyyn DevOps-kypsyysmallin. Työn lopuksi muodostetaan arvio projektin DevOps-kypsyydestä ja DevOps-kehitysmalliin siirtymisen ongelmakohdista.

Tutkimuksessa tehtyjen havaintojen perusteella projektin suurimmat haasteet ovat projektin suuri koko ja kompleksisuus, ongelmat projektinhallinnassa, ajoittaiset ongelmat kommunikaatiossa järjestelmätoimittajan kanssa, ohjelmiston huono laatu ja puutteet järjestelmätoimittajan ohjelmistokehitysprosessissa. Haasteista huolimatta yritys on onnistunut parantamaan osaa projektin osa-alueista, kuten automatisoitua osan ohjelmiston käyttöönotosta, luomaan yksinkertaisen monitoroinnin ja neuvottelemaan ohjelmistokehitys ja -testauskäytännöistä järjestelmätoimittajan kanssa.

Asiasanat: DevOps, ketterä ohjelmistokehitys, jatkuva integraatio, organisaatiokulttuuri

Contents

List of Figures

List of Tables

1 Introduction

1.1 What DevOps Is?

DevOps has been a hot topic for the past couple of years. The DevOps paradigm addresses the split and barrier between developers and operations personnel (Wettinger et al., 2014). The developers include software developers, testers, and quality assurance personnel. The operations include system administrators, database administrators, network technicians, and other roles that assist with the production infrastructures. (Liu et al., 2014). Although the main purpose of DevOps is clear and straightforward, there are many interpretations what DevOps actually means (Smeds et al., 2015). In his popular blog post, Willis (2010) mentions that the situation is much like early days of cloud computing. Many of the questions asked are not yet answered. The situation has not considerably changed five years after the blog post. There are still questions without answers regarding DevOps.

Smeds et al. (2015) conducted a literature review and analysed the definitions of DevOps proposed in the literature. The authors selected 27 publications for their literature review. Academic publications databases like EBSCO and Springer Link were used as a source. Judging from the sources, the literature review was solely based on academic research papers. The authors ruled out blog posts as a quality precaution. However, the authors admit that blog posts contain a significant amount of information related to DevOps. As indicated by Smeds et al. (2015), there is a limited amount of academic material related to the topic. Therefore, this thesis selects a different pathway and joins

together information from academic research and professional literature.

In blog posts regarding the topic, two major views prevail, other sees DevOps as a job position requiring a mixture of development and operation skills. The opposing view is that DevOps is much more than only a job title. According to Smeds et al. (2015), the existing literature focus on different aspects of DevOps. Some authors emphasize the culture aspects of DevOps without addressing the specific technical context. The definition of DevOps should also include the engineering practices influenced by the cultural aspects.

DevOps inherits from Agile System Administration and Enterprise Systems Management movement and it has strong links with Agile and Lean approaches. It can be seen as an extension of Agile principles beyond the boundaries of development work to the entire delivered service. In the late 2000s, agile development was moving from niche to common practice. This development turned to thinking about Agile System Administration especially in Europe. This movement was focused on the process and analogies from Kanban and Lean manufacturing processes to IT system administration. In 2009, Patrick Debois and Andrew Shafer met and coined the term ”DevOps”. After that, Debois held the first DevOpsDays event in Ghent, Belgium. (theagileadmin.com, 2011)

At the time DevOps was born, enterprises ran agile and lean development cycles, but their operations resembled waterfall processes (Willis, 2010). This kind of mismatch in workflows will probably cause problems in delivering quality software. According to Liu et al. (2014), in traditional division between development and operations team, development team strives for change and updates, whereas operations team strives for stability. The teams have isolated goals which will complicate their cooperation. Instead of optimizing the process as a whole, these two separate teams will optimize their own processes. Developers adopt Agile methodologies to accelarate the creation of new features, whereas operations use practices like ITIL (Information Technology Infrastructure Library) to maintain stability and enhance performance. (Hüttermann, 2012). Collaboration across the value chains is fundamental part in DevOps. Integrating traditionally separated teams

allow rapid and frequent delivery of services and products. (Liu et al., 2014)

In the ideal situation of DevOps, development and operations personnel are working in harmony without organizational barriers. Their common goal is a seamless and repeatable software delivery process. DevOps is a set of tools and methods in order to streamline this process. Despite some authors see DevOps as an organizational structure, it is not a way to get rid of operations or development team. The harmony of development and operations is essential in DevOps. (Swartout, 2014).

To summarize, providing a clear and unambiguous definition for DevOps is difficult, because it consists of several overlapping aspects. It can also be observed from several different viewpoints. The research firm Gartner provides a following definition: ”DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture) and seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology — especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective.” (Gartner, 2016). The definition emphasizes that DevOps is a change in culture and ways of working, and that technological means are used only in the implementation stage to support this change. So, they are not an end in itself.

1.2 Structure and Research Questions

This thesis focuses on DevOps adoption. However, before studying adoption, effort is made to define, describe and observe DevOps from different perspectives. Due to this, the first research question (RQ1) is defined as: ”What is DevOps and what does the term contain?”. The questions are answered by studying literature on topic and earlier studies related to the topic.

Answers to the first question create a basis for second research question (RQ2) which is:

”Which factors should an organization consider when adopting DevOps?”. The questions are first answered based on literature. Then, findings are used in a qualitative case study for creating questions for the interviews and deciding which aspects in the organization could possibly be problematic for the DevOps adoption.

Given these two research questions, the goal of this thesis is to first study what the DevOps movement is all about. After that, to study which factors an organization should take into account when adopting DevOps. More specifically, which factors contribute negatively and which positively to DevOps adoption.

The remaining of this thesis is structured as follows. Chapter 2 continues to define what DevOps is and which aspects it contains. DevOps is observed through the core values of DevOps movement. These values are: culture, automation, measurement, and sharing. In this chapter, common technical tools used in DevOps are described and introduced.

DevOps adoption in an organization is covered in Chapter 3. The chapter identifies which aspects contribute positively to organization’s DevOps adoption. Likewise, aspects contributing negatively to DevOps adoption are identified. The aspects are divided into capabilities, enablers, and impediments. At the end of the chapter, two DevOps maturity models are presented and compared.

Chapter 4 introduces the case study part of the thesis and the research methods used in the case study. The chapter defines the purpose and objectives of the case study and presents the studied organization and system. Chapter 5 continues the case study part, presents the results and provides a DevOps maturity estimation of the project.

At the end of the thesis, Chapter 6 reflects the thesis in a more discursive way and analyzes what could have been done differently in the thesis. Chapter 7 provides a conclusion for the thesis. It sums up the study and presents implications, limitations and future work.

2 CAMS: Core Values of the DevOps Movement

2.1 Introduction

Regarding to DevOps Dictionary (2015), CAMS is an acronym describing the core values of the DevOps Movement: Culture, Automation, Measurement, and Sharing. The acronym was coined by Damon Edwards and John Willis in 2010. Minick (2015) mentions that CAMS has been extended repeatedly. Jez Humble added an L, standing for Lean, to form CALMS (Willis, 2016). Oehrlich (2015) extended the acronym even further by adding additional S, standing for Sourcing. Hereby, the acronym CALMSS was formed. According to Oehrlich (2015), additional S was added because there should be a solid sourcing strategy to support DevOps. Hence, DevOps can be observed from many different perspectives and it can be extended to cover various aspects in software development. To summarize, there is no unambiguous way to define core values or aspects in DevOps.

For the sake of simplicity, the acronym CAMS is used to define the core values of DevOps in this thesis. The acronym also provides structure for this chapter. Each section presents one core value and its role in DevOps culture. In addition, certain practices and tools related to these core values are introduced.

2.2 Culture

2.2.1 Importance of Culture

Willis (2010) underlines the importance of culture by stating: ”People and process first. If you don’t have culture, all automation attempts will be fruitless.” Walls (2013) agrees by stating that tools are not enough to create the collaborative environment many of us refer to now as DevOps. She stresses this by adding that putting together a new set of tools is simple when compared with changing organizational culture. Creating a collaborative environment for DevOps is a huge challenge, because it requires assessing and realigning how people think about their teams, businesses, and the customers. DevOps is as much about culture as it is about tools. Using DevOps-friendly tools and workflows can help a team to work in a more DevOps manner. However, creating a culture that is supportive of the DevOps movement ideals is crucial. (Walls, 2013)

2.2.2 Characteristics of DevOps Culture

According to Walls (2013), a culture suitable for DevOps is created through lots of discussion and debate. DevOps Dictionary (2015) compares pathological and healthy organization by stating that in pathological organization, it is unsafe to ask other people questions or to look for help outside of official channels. Whereas in healthy organizations such behaviour is rewarded and supported with inquiry into why existing processes fail. Walls (2013) makes a comparison between traditionally siloed technical teams and DevOps approach. The former interacts through complex ticketing systems and ritualistic request procedures. While the latter talks about the product throughout its lifecycle. The overall focus is on the product, not gathering political power.

DevOps Dictionary (2015) points out that DevOps culture has roots in Agile and Lean. Lean and Agile are both focused on people first, systems second, and heavily cross-train or work in cross-functional teams. In DevOps, collaboration is considered as a cure to

thinking in silos which causes conflict and mismatch between development and operations. In addition to collaboration, Walls (2013) highlights respect and trust as essentials in building a DevOps culture. Members of the organization do not actually have to like each other, but everyone needs to recognize each other's contributions and treat others well. Respectful discussion and listening others' opinions is a learning experience for everyone. Operations must trust that development is doing their best because it is the best for the success of the product. Management must trust that operations is going to give objective feedback and metrics after the next deployment. If some part of the organization do not to trust some other part, even the most cutting-edge tools would not matter.

Regarding to Minick (2015), other cultural emphasis is moving towards experimentation, constant learning, and improvement. The author mentions this as Kim's (2016) third way of DevOps. According to Kim (2016), this kind of cultural aspect needs two things. The first is continual experimentation, taking risks and learning from failure. The second is to understand that repetition and practice are the prerequisite for mastery. These are both needed because the first ensures that the organization pushes to improve. This can even mean going deeper into the danger zone that organization have ever gone. The mastery of skills is needed when the organization needs to retreat from the danger zone.

According to Minick (2015), culture of experimentation, constant learning, and improvement works best when there is ample feedback. If there is no feedback, there is no real ground for learning. Sections 2.3 and 2.5 present technical concepts like continuous integration and monitoring, which help to increase the level and detail of feedback, and also to fasten the feedback loop. However, technical tools and concepts are only a part of DevOps. The organization should embrace a culture where these tools are used in an effective manner. Minick (2015) highlights that feedback must cross the traditional silo lines. The author gives an example: "Awesome production monitoring that had lived in Ops land, is radiated back to development".

2.3 Automation: Foundation

2.3.1 Role of Automation

DevOps Dictionary (2015) mention that automation is perhaps the most visible aspect of DevOps and many people focus on productivity gains as the main reason to adopt DevOps. However, in addition to saving time, automation is also used to prevent defects, create consistency and enable self-service. Minick (2015) adds that feedback needs automated deployment of changes, and automated tests or production monitoring that validates them. According to him, automation is what makes DevOps possible. He sums up automation by stating: ”DevOps isn’t about automation. Automation is just the natural result of DevOps”.

Willis (2010) mentions that automation is one of the places to start once organization has understood its culture. According to the author, tools for release management, provisioning, configuration management, systems integration, monitoring and control, and orchestration are important pieces in building automation. These tools and their roles in an automated pipeline are presented in the next section.

2.3.2 Release Automation

Automating common tasks in building, testing, and releasing software helps to increase efficiency and setup a reproducible process that can be implemented by toolchains. Provisioning and deployment of the virtual machines, middleware and application code can be also automated. Automating these tasks ensures repeatability. One key goal of automatic releasing is to reduce the risk of any individual release. (Hüttermann, 2012)

Hüttermann (2012) advises doing small releases often as opposed to doing big releases seldom. This is because frequent deployments to production will help keep things simple. In addition, individual changes will be more focused. The process of deployment will be practiced constantly which reduces the risk of deployments.

Continuous deployments will help to identify problems in processes and toolchains

earlier and they can be optimized accordingly. When change size is small, learning about the root causes of production incidents and getting system back up again is easier because troubleshoot scope is limited. Once error is uncovered, it can be fixed. That makes a total rollback unnecessary.

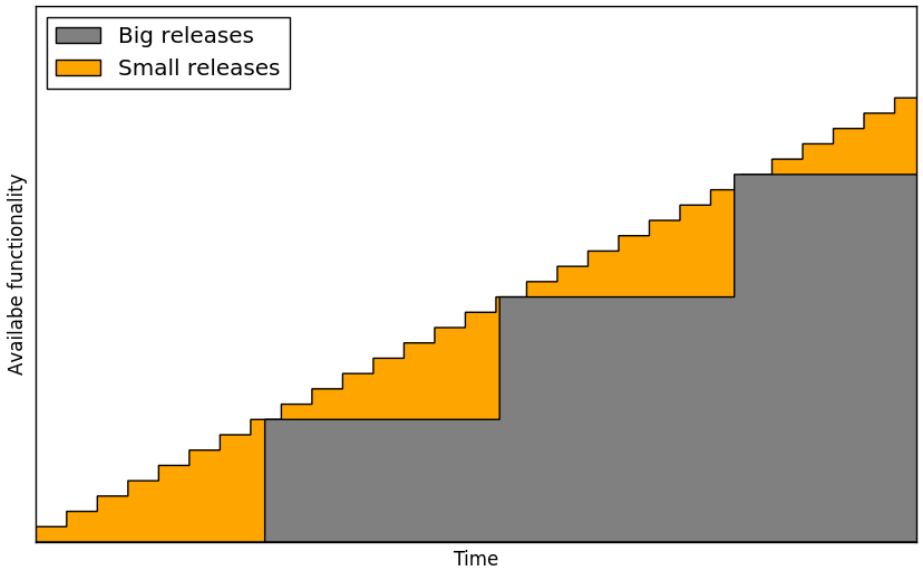

Figure 2.1: Small releases deliver value earlier. Adapted Hüttermann (2012)

Figure 2.1 illustrates how choosing small releases eventually deliver the same amount of functionality compared with big releases. However, choosing small releases will deliver more functionality more quickly. This means that software will return value quicker. (Hüttermann, 2012)

Hüttermann (2012) reminds everyone that automation activities should be driven by business instead of technical considerations. These activities must result in concrete benefits. There is no sense in automating for the sake of automation. Automation is performed to gain fast feedback and it should make humans more important, not less important. Consequences of automation, like marginal costs, should always be kept in mind when planning to automate releasing.

2.3.3 Examples of Deployment and Release Strategies

Improving and accelerating delivery is one building block of DevOps. Decoupling deployment and release support this building block. Deployment and release decoupling can be done using several strategies. Branch by abstraction, feature toggles, dark launching, blue-green deployment, and canary releasing are examples of these strategies. (Hüttermann, 2012). In the following, these are presented with more details.

The *branch by abstraction* makes large-scale changes to the system incrementally. Initial step to implement this strategy is to create an abstraction over the part of the system that will be changed. After the abstraction, the rest of the system is refactored to use the abstraction layer. Implementing the change is then continued iterating between adding new code and removing the implementation. The abstraction layer delegates to the old or new code, as required. After the old implementation has completely been replaced, the abstraction layer is removed. (Hüttermann, 2012)

The *feature toggle* delivers the complete code to production but uses data-driven switches to decide which feature is made available during runtime. Configuration files are often used to enable data-driven switches. With the feature toggle, the team can develop on the same development mainline without the need using branches and ship the complete code to production. The disadvantage of feature toggles is that production code contains parts that are not relevant to that specific release, because some of the features are faded out. This nonrelevant code may influence other code parts or even introduce errors. (Hüttermann, 2012)

The concept of *dark launching* is to deploy first versions of functionality into production for a certain subset of users before releasing the functionality to all users. Bugs can be found more easily and before the release is available to all users. It provides an approach to remediate in a low-risk way. Only a few users will experience the problems in early versions of the feature. Incidents can be addressed without a complete heavyweight rollback, just by switching off the feature, using for example feature toggle, or by changing

a router setting. (Hüttermann, 2012)

The core of *blue-green deployment* strategy is to deploy a new version of the application side by side with the old version. This strategy makes possible to switch versions back and forth just by changing load balancer or router settings. The strategy ensures that two similar production environments exist. At any one time, one of them, green environment for instance, is live. A new version of the software is then tested in the blue environment. When the software works as excepted in the blue environment and the tests are successful, the router is switched to direct all incoming requests to go to the blue environment. After this, the green environment is out of production and can be used to prepare the next release. (Hüttermann, 2012)

Canary releasing is a variation on blue-green deployment. It is applied when running a cluster of servers. Rather than upgrading a whole cluster to the latest version all at once, the deployment is done incrementally. The new release is first deployed to a subset of production systems, to which specific users or groups are routed. If the deployment is successful, the release can be deployed to more users and finally to all available users. (Hüttermann, 2012)

2.4 Automation: Tools

2.4.1 Introduction

DevOps tries to remove the barrier between development and operations personnel. According to Hüttermann (2012), using unified toolchains across the whole development and release process is one part of achieving this goal. In addition, automation helps to make this whole process transparent.

In this section, essential tools for DevOps are presented at high-level. Technical details are left out intentionally and the focus is to explain the main functions of different tools and how they are related to each other creating an interoperable whole. At the end of this

section, an example use case how the different tools interoperate is shown.

2.4.2 Continuous Integration

Continuous Integration, more commonly known as CI is a software solution which allows running automated scripts triggered by certain events like a commit to source control or once an hour, for example (Swartout, 2014). Usually developer works on a small portion of the code at the time, and run unit tests only for that portion. Sometimes it is nearly impossible for the developer to test his local changes in production like environment. It is possible that the change the developer made to one part of the code leaves the overall application in a non-working state. (Mikita et al., 2012).

CI is a method of ensuring that the software being developed builds correctly and integrates with the rest of the platform. This ensuring process is done continuously, hence the name, preferably on each commit. CI job contains a list of activities which need to be run successfully every time the job is executed. (Swartout, 2014).

A successful continuous integration process relies on version control and ability to build and test automatically. The commits to version control needs to be small in size, because that means there is a smaller change of breaking the overall build and less change of conflicts with other developers. The need for automated build is self-explanatory, because the whole system or component needs to be rebuilt after every commit. Comprehensive set of tests are run after every commit to ensure functionality and that requirements are met. In addition to these technical requirements, every member of the team must fully commit to using the continuous integration tool and practices. Without this commitment, the CI plan will likely be unsuccessful. (Mikita et al., 2012)

An example use case could be the following: get the latest version from source control, compile source code to executable, deploy the binary to a test environment, get the automated tests from source control, and run the tests. If there are no errors, the CI job completes and reports success. In case of errors, the CI job fails and provides detailed

feedback about the failure. (Swartout, 2014)

CI provides a complete audit trail and helps to distribute information accurately. For example, it can send an email to the developer who committed the change which caused the build to break. CI is a powerful tool, which helps to reduce the risk caused by doing the building, testing, and deploying manually. It also frees developers to do actual development work instead of doing the same manual tasks over and over again. (Swartout, 2014)

Jenkins (Jenkins, 2015), Bamboo (Atlassian, 2015), and TeamCity (JetBrains, 2015) are examples of Continuous Integration software. Jenkins is an extensible open source continuous integration server and it consider as de facto solution. It is Java-based and mainly used via web interface.

2.4.3 Automated Testing

Automating the tests and running them constantly provides confidence that the system is working correctly after a change to the code or environment. If the tests are to run over and over again with the same results, developers can trust that the system still works. In case that tests fail at some point, it is likely that the last change broke something. In other words, there is a clear starting point and scope for the debug process. It possible to do all of the testing manually, but that can be slow, error-prone, inconsistent, and not always fully repeatable. (Swartout, 2014)

Automated testing should take place at multiple levels including unit tests, component tests, and acceptance tests. Running these tests every time a change is made to the application or configuration is achieved by the Continuous Integration and source control. An unit tests validates a small piece of code. At code level this small, piece is typically a single class or method. They are always written and maintained by the developer. Unit tests are run in isolation, so they do not involve making database calls or use of any other external resources. To cover this, mock data is used in testing. Component tests are similar to unit tests, but they are larger and involve the usage of external resources. Due to this,

they are slower to run. (Mikita et al., 2012)

Acceptance tests ensure that acceptance criterion of a story is met and they can test attributes of all kinds. The tests should be written by business users, because they have more comprehensive domain knowledge compared with the developers. Acceptance tests consist of two categories: functional tests and non-functional tests.

Functional tests are the most important tests, because they answer from the developer's point of view to the questions "Am I ready?" and "Did I deliver what customer wanted?". They would also determine if the change broke something else in the whole system. Non-functional test tests the qualities and attributes of software. Performance, security, and capacity are examples of those kinds of qualities and attributes. It is important to consider these tests when designing a test suite. Correctly functioning system will not be useful if its performance is not at acceptable level or if it contains security vulnerabilities. (Mikita et al., 2012)

Automated tests should be planned thoroughly and systematically. Test coverage, test environments, and test data need to be defined in such a way that the test results can completely be trusted. Defining the tests can sometimes be a complex task, but the main instruction is to keep everything simple. At first, tests should cover primary cases. These cases can be refined or more cases can be added later. Automated tests are usually executed using continuous integration software, which was introduced in the previous subsection. (Swartout, 2014)

Most of the programming languages have their own frameworks for unit testing like Java has JUnit and C# has NUnit. Unit tests are run automatically by Continuous Integration software. Tools like Selenium or Robot Framework can be used for automating acceptance tests. Selenium automates browser interactions and its primary use case is automating web application testing. It has support for many browsers and operating systems and it can be controlled by many programming languages and testing frameworks. (Selenium, 2015). Robot Framework is a generic test automation framework, which has easy-to-use tabular

test data syntax. Its testing capabilities can be extended by test libraries implemented either with Python or Java. (Robot Framework, 2015)

2.4.4 Source Control

Source code should be stored in source control, sometimes referred to as SCM or version control system. In source control, code is versioned meaning that there is a history of every change from the start of the project. It is also easily available to everyone who has access to the system, secure and usually backed up so nothing is lost in case of hardware failure or other problem. (Swartout, 2014). For example, SVN (Subversion, 2015), Git (Git, 2015), and Mercurial (Mercurial, 2015) are one of the most common source control software. (Mikita et al., 2012)

Source control usage should not be restricted to source code, but if any part of the platform is represented and stored as a text file, it should be stored within source control. This includes system configuration, start-up scripts, server configuration, and network configuration for example. (Swartout, 2014). Mikita et al. (2012) also addresses the importance of storing every single artifact related to the creation of software to the source control. This allows an easy re-creation of the testing and production environments, and the development team can always roll back to the last previously known good state, if any modification to the system introduces errors. Also, the introduction of a new team member is easier because all of the needed source code, tools, scripts and configuration files are stored in one, easily accessible place.

A source control solution is an essential part of DevOps toolset, because it helps to keep changes small, frequent, and simple. When changes are small and frequent, change impact is small, so the risk that a change breaks something is reduced. This is a good practice, because it reduces complexity, maintains quality, and provides understanding how a breaking changes can can affect the system. Merging is also easier, because there is only small amount of code to merge together. (Swartout, 2014).

Source control systems offer a possibility to add a small message to every commit that is made. This small message gives an idea of what the purpose of the change is. Commit messages provide a sort of documentation and history for the project, because they explain what was done and why. The messages should be concise and descriptive. (Mikita et al., 2012). This is another reason for keeping commits small and simple, because it is easier to provide a short and a descriptive commit message for a small commit.

2.4.5 Environment Management and Provisioning

Applications depend upon certain operating system and other applications in the operating system. Application might require certain port to be open for functioning correctly. The worst possible way for environment management is to manage them by hand and an ad hoc basis. Using this method, there is no record of the last working configuration if a problem occurs with the current configuration. Hence, there is no method in order to do rollback. (Mikita et al., 2012)

The process of environment management should be automated. This removes the possibility that there is only one person who knows how to setup a new environment. In addition, it provides a method for rollbacking to the last working configuration. It allows also to create production-like test environments for development and testing use. Configuration specifications can be pushed to version control. From there it can be pushed out to various environments using tools like Puppet (Puppet Labs, 2015) or CFEngine (CFEngine, 2015). User access levels, installed software, and other details can be specified in configuration specifications. (Mikita et al., 2012). In addition to tools mentioned in this paragraph earlier, Chef (Chef, 2015) and Vagrant (Vagrant, 2015) are other common tools used for environment management and automation.

Automated provisioning can be a part of the deployment process if the platform runs completely off virtual infrastructure. It can be quite complex and painful to implement, but it is extremely useful and powerful. After all automated provisioning means ability

to programmatically tell the system the wanted specifications, operating system, and configuration, and output the result to the other end. Process of automated provisioning could be the following: provision a server the software needs to run on, deploy the software onto the server, install the software, add the server to the platform, and start using the server. (Swartout, 2014)

2.4.6 Dependency Management

Most of the software projects have dependencies, which can be internal components or third party libraries. The main goal of dependency management is to ensure consistent and repeatable builds. There are two main approaches to achieving this. First one is to check all of the dependencies to version control, second is using dependency management tools like Maven (Apache Maven Project, 2015) or Ivy (Ivy, 2015). These tools will transitively resolve the dependencies of a project and ensure that there are no inconsistencies in the dependency graph of a project. (Mikita et al., 2012)

Another side in dependency management is the usage of a personal artifact repository. Components which are used in multiple projects within an organization should be stored to an artifact repository. An artifact repository helps to divide a project into components, which provides more freedom to optimize builds and deployments. Sonatype's Nexus (Sonatype, 2015) and JFrog's Artifactory (JFrog, 2015) are examples of the personal artifact repositories. Both of these tools have open source and commercial versions. (Mikita et al., 2012)

2.4.7 Monitoring

Impact of any change is easy to see and there are no hidden surprises, if all of the environments are monitored. Monitoring is crucial part of DevOps, so that both development and operations personnel can see what is going on and can provide assistance if there are problems. (Swartout, 2014).

Nagios, Collectd, and Munin are examples of monitoring tools, which Hüttermann (2012) mentions by name. Nagios is a tool that provides an instant awareness to organization's mission-critical IT infrastructure (Nagios, 2015). Collectd gathers statistics about the system it is running on and stores this information. Those statistics can then be used to find current performance bottlenecks and predict future system load. (Collectd, 2015). Munin is a networked resource monitoring tool that can help analyze resource trends and "what just happened to kill our performance?" problems (Munin, 2015).

Data from different tools should be integrated into unified views from where the most important information can be seen easily with one look. Monitoring enables engineers to see how the software behaves in real time with real users. Access to monitoring views should not be restricted only for technical personnel, but it should be visible everyone in the business. That is how they can see how the platform is performing. (Swartout, 2014)

2.4.8 Deployment Pipeline

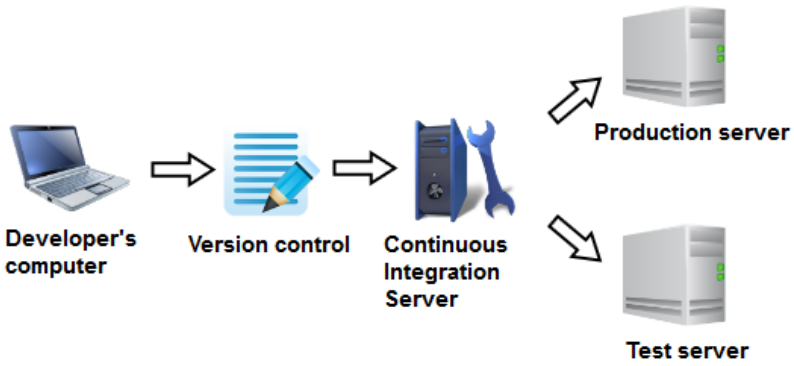

Figure 2.2: Simplified presentation of deployment pipeline

Figure 2.2 shows a simplified version of deployment pipeline using tools described along this section. The continuous delivery process starts when the developer pushes code changes to the version control system, which stores the versioned project artifacts. The continuous integration server watches the version control repository for changes and starts its predefined job when a change occurs. This job usually consists of testing, building, and

deploying the software.

Unit and component tests are run before the build. If these tests succeed, the continuous integration server starts building the software. After successful build, software is deployed to test environment and the automated user acceptance tests are run. In case these tests are successful, software is deployed to the production environment.

If some of these steps fail, the continuous integration server informs people involved. The server can be configured in many ways, it can for instance send an email to the developer who made the commit which broke the build. It also provides a clear audit trail and logs, which helps to fix the issue.

2.5 Measurement

2.5.1 Introduction

Constant learning and experimentation are characteristics of DevOps culture. These characteristics require feedback, which can be established with measurement. Willis (2010) highlights this by stating: ”If you can’t measure, you can’t improve”. The author adds that a successful DevOps implementation will measure everything it can, as often it can.

Technical monitoring tools were briefly presented in the previous section. Using monitoring tools allow sharing real-time status of the system and its environment across developers and operations staff. However, even the most state-of-the-art monitoring tools will not matter if the measured metrics are not relevant. According to Hüttermann (2012), selected metrics should be meaningful enough to aid all participants. The next subsection presents and describes metrics in the DevOps context.

2.5.2 Metrics

According to Riley (2015) metrics are needed for measuring the success of organization's DevOps program. In addition, they provide a way to find out how it can be improved, modified, or extended. Hüttermann (2012) warns that metrics which try to summarize and aggregate highly complex dependencies into single numbers should not be used. This is because counting leads to the illusion that something can be understood because it can be quantified. Numbers can be very misleading and counting these kind of metrics can lead to twisted incentives.

There are numerous things to measure in DevOps. Riley (2015) presents four key metrics used by Puppet Labs, whose primary product is the configuration management tool Puppet. These four metrics are: Deployment (or change) frequency, change lead time, change failure rate, and mean time to recover (MTTR). According to Riley (2015), good DevOps metrics should at least cover these four areas along with other key indicators of performance which matter to the organization.

According to Riley (2015), in a DevOps environment deployment frequency can be a direct or indirect measure of response time, team cohesiveness, developer capabilities, development tool effectiveness, and overall DevOps team efficiency. The author adds that with fast feedback and small batch development, updated software can be deployed even several times per day.

The term 'change lead time' refers to the time from the start of the development cycle to deployment (Riley, 2015). It can be used to measure the efficiency of the development process, complexity of the code and the development systems, and team and developer capabilities. Hüttermann (2012) refers to a similar metric with the term 'cycle time', but mentions that it is useful only when definition of done is the point where features have been developed, tested, and shipped to the customer.

For frequent deployments to have value, the failure rate needs to be low. Change failure rate metric is used to keep track of the overall success of deployments. The metric

should decrease over time as the experience of DevOps teams or developers increase. An increasing failure rate is a good indication of problems in the overall DevOps process. (Riley, 2015)

Mean time to recover (MTTR) metric refers to the time from a failure to recovery from that failure. Generally, it is a good measure of team capabilities and it should also decrease over time. Code or platform complexity, new features, and changes in the operating environment can affect MTTR. (Riley, 2015)

2.6 Sharing

According to DevOps Dictionary (2015), key to the success of DevOps at any organization is sharing the tools, discoveries, and lessons. Regarding Minick (2015) the ”CAMS” concept of measurement and sharing point to feedback. The author simplifies this by stating that lots of information is gathered and then it should made sure that people see it, so they have an opportunity to learn from it. Cultural characteristics like respect, trust, and constant learning and improvement support sharing. DevOps Dictionary (2015) defines sharing in organization as: ”finding people with similar needs across the organization, new opportunities to collaborate can be discovered, duplicate work can be eliminated, and a powerful sense of engagement can be created among the staff.”

In addition to sharing ideas and innovations inside organization, there should be sharing outside organization, according to DevOps Dictionary (2015). Sharing tools and code in the community helps to get new features implemented in open source software quickly. Conference participation leaves staff feeling energized and informed about new ways to innovate. Sharing outside organization feels like a natural aspect of DevOps, given its birth story and its first steps.

2.7 Chapter Summary

This chapter presented acronym CAMS, which describes core values of DevOps: Culture, Automation, Measurement, and Sharing. DevOps needs a collaborative organizational culture which is supportive of the movement ideals. Creating that kind of culture is crucial. Key characteristics of DevOps culture are collaboration, respect, trust, constant learning, feedback, and experimentation.

Automation boosts productivity by saving time. In addition, it is also used to prevent defects, create consistency, and enable self-service. Tools for release management, provisioning, configuration management, and automated testing are examples of pieces in building automation. Automation add up to feedback by creating constant metrics about deployment changes, automated tests, or production monitoring.

Successful DevOps implementation will measure everything it can, as often it can. Measurement creates constantly updating metrics which enables feedback. Usage of monitoring tools allow sharing real-time status of the system and its environment across developers and operations staff. However, measured metrics should be selected carefully and they should be relevant.

Sharing should be supported and encouraged by the organization culture. Organization should share its tools, discoveries, and lessons. By doing that, organization can discover new opportunities to collaborate, eliminate duplicate work, and engage staff. Innovations and ideas should also be shared outside the organization. This can be for example done by taking part in open source software development and participating in conferences.

3 DevOps Adoption

3.1 Introduction

Adopting DevOps may require that an organization introduces process, personnel, and technological changes and innovations. Implementing changes like these in organization is seldom an easy or straightforward task. As in any software process improvement initiative, the path to successful DevOps adoption is unique to each organization. Regardless the unique nature of this improvement initiative, it is possible to learn from challenges experienced during DevOps adoptions in order to plan future DevOps adoption initiatives. (Smeds et al., 2015)

In their study, Smeds et al. (2015) conducted a literature review about factors which contribute to organization's DevOps adoption. They divided these factors to DevOps capabilities and enablers. Enablers were further divided to cultural enablers and technological enablers. In the same study, Smeds et al. (2015) identified impediments to DevOps adoption by interviewing employees of an international IT company with a long history and over 1,000 employees. At the time interviews were conducted, company's DevOps adoption process was at an initial stage. Authors emphasize that adopting DevOps may not be trivial for large organizations with complex service requirements. Impediments the authors studied can complicate DevOps adoption from the perspective of capabilities, cultural enablers, and technical enablers.

At the time of writing this thesis, Smeds et al.'s (2015) was the only study found which

covered DevOps adoption from wide-ranging organizational viewpoints and contained a case study. Therefore, this chapter is mainly based on Smeds et al.'s (2015) study.

In this chapter, Smeds et al.'s (2015) division into capabilities, enablers, and impediments regarding DevOps adoption is used to structure the chapter. Smeds et al.'s (2015) findings are incorporated with findings from other studies. At the of this chapter, various DevOps maturity models are studied and compared.

3.2 DevOps Capabilities

Regarding to Smeds et al. (2015), capabilities define processes that organization should be able to carry out. Capabilities regarding DevOps comprise the basic activities in software and service engineering: planning, development, testing, and deployment. Smeds et al. (2015) highlights that the three most essential factors regarding these basic activities are: continuous operations, feedback, and ability to recover from failures. Facilitating these factors in the basic activities will create capabilities for DevOps.

3.2.1 Continuous Operations

Smeds et al. (2015) define the term continuous in this context as in small increments and without delay. An example for capability like this is continuous deployment. It allows an organization to deploy a new feature as soon as they are successfully integrated and tested. For this to be possible and efficient, organization should have automated testing and deployment tool chain and streamlined the collaboration between developers and operations. (Smeds et al., 2015)

3.2.2 Feedback from Monitoring Data

In DevOps context, feedback is understood as using data collected from service operation in planning and development. The feedback data contains service performance data and

data on user interaction with the service. Infrastructure monitoring and user behaviour monitoring capabilities cover this data collection. Data collection provide feedback loops for the planning and development processes. These feedback loops are used to improve and optimize the service. (Smeds et al., 2015)

3.2.3 Service Recovery

Covering the last factor, a DevOps organization should have the capability to recover from service failures without delay. Service failures can be caused by the service infrastructure or by software defects. For immediate failure detection, the organization should have the necessary monitoring infrastructure. In addition to detection, there should be contingency plan for reacting to the failures. (Smeds et al., 2015)

3.3 DevOps Enablers

Smeds et al. (2015) divide DevOps enablers to two categories: cultural enablers and technical enablers. Cultural enablers are traits that DevOps team and organization should exhibit. The technological enablers stress the need for automating tasks. Walls (2013) emphasizes the cultural aspect of DevOps adoption in her book. Tools are not enough to create the collaborative environment that many of us refer to now as DevOps. Creating collaborative environment requires assessing and realigning how people think about their teams, the business, and the customers. Putting together a new set of tools is simple when compared with changing organization culture. (Walls, 2013)

3.3.1 Cultural Enablers

Cultural enablers will positively contribute to the DevOps capabilities (Smeds et al., 2015). Similar to cultural enablers found in Smeds et al. (2015) study, Walls (2013) lists key cultural characters for creating DevOps culture. From these two, the following list of

cultural aspects contributing positively to DevOps adoption can be made:

- Open and effortless communication
- Shared responsibilities, common goals, and extensive collaboration
- Supportive working environment

A team taking DevOps approach discusses requirements, features, schedules, and resources throughout the lifecycle of a product. Monitoring data like production and build metrics mentioned in DevOps capabilities supports open communication. This data should be prominently displayed and available for everyone. In addition to monitoring data, documentation helps to foster open communication. In organization culture incorporating open and effortless communication, every team member is able to ask questions from other team members and expect to get an answer or help in finding one. (Walls, 2013)

The team should have a common goal, building a solid and functioning product. Shared responsibilities, common goals, and extensive collaboration can be promoted by setting a uniform incentive for the team. For example, development should not be rewarded for writing lots of code. Likewise, operations should not be punished if the code does not run as expected in production. The team should be rewarded when product is excellent. (Walls, 2013)

Smeds et al. (2015) mentions supportive working environment as a cultural enabler. Walls (2013) defines this further by mentioning respect and trust. All team members should respect each other. Everyone should recognize the contributions of everyone else and treat their team members well. Trust is an essential component of achieving a DevOps culture. Tools will not matter if any part of the team does not trust another part of the team.

Walls (2013) mentions that team managers or leads have a role in building supportive working environment. Building respect in a team derives from how the manager or lead resolves interpersonal conflicts among team members. Bad fights like personal attacks or name-calling should be dealt with from a managerial standpoint. Destructive

behaviours should not be ignored. In non-constructive conflict situations, manager or team lead should mediate a compromise if one is warranted. Trust in a team is build over time. Positive changes in communication, responsibility, and respect builds an open collaborative environment.

In addition to above-mentioned three cultural aspects, Walls (2013) raises the need for executive sponsorship and technical leadership in DevOps adoption. If high-level executives support the adoption, necessary organizational changes like compensations and incentives can be facilitated more easily. Having an executive sponsor is also helpful when spreading the good news about cultural adjustments. The role of technical lead is important, it is a combination of evangelist, tools expert, and process subject-matter expert. Technical lead can be one person or team of people and they will help to facilitate tool, behaviour, and workflow changes.

Regarding to Swartout (2014) companies should build a culture throughout the business where innovation is recognized as a good and worthwhile thing rather than a risky way of advancing a product. The author adds that there should be some room for investigation and experimentation. However, employees must understand that with freedom comes responsibility, ownership, and accountability for the new things that has been built or produced. Smeds et al. (2015) agrees on this by stating that innovation is promoted by allowing both teams and individuals to experiment and learn from their successes and accept failures as a learning experience.

Whereas the mentioned behaviours will contribute to the DevOps capabilities in a positive way, respectively certain behaviours will negatively contribute to the DevOps model. These kind of behaviours are: blaming others for failures, showing disrespect towards fellow employees, and considering only personal work performance. (Smeds et al., 2015). Later section presents additional matters which negatively affect cultural enablers.

3.3.2 Technological Enablers

Technological enablers emphasize the need for task automation. Task automation is beneficial because it decreases the amount of errors in the system. It also shifts the focus of the employees from manual error-prone repetitive tasks to creative and productive tasks. Automation supports DevOps capabilities by making the software and service development process more streamlined and stable while allowing employees to be innovative and productive. (Smeds et al., 2015). In addition to reducing errors, software company Eficode's (2015) report highlights cost reduction by automation. The author states that testing and releasing software continuously and automatically reduces errors and costs resulting from infrastructure management and the development work itself.

Build automation, test automation, infrastructure automation, and configuration management are examples of technological enablers (Smeds et al., 2015). These concepts and tools were presented more closely in Section 2.3. Build automation can be achieved with continuous integration and dependency management tools. Similarly, test automation can be achieved with automated testing tools in various testing phases like unit testing and acceptance testing. Infrastructure automation and configuration management are related to tools presented in Section 2.4.

Lwakatare et al. (2015) mentions that one approach to address the manual process is depicted in the concept of "Infrastructure as Code" (IaC). Regarding to Lwakatare et al. (2015) the IaC concept is used to describe the idea that almost all actions to the infrastructure can be automated. Configuration management tools are central to facilitating IaC concepts. The concept underlines need for developing automation logic for deploying, configuring and upgrading software and infrastructure repeatedly and quickly, particularly in a cloud environment (Lwakatare et al., 2015).

3.4 Adoption Impediments

3.4.1 Impediments Affecting Capabilites

Regarding to Smeds et al. (2015) the impediments affecting capabilities are: unclear definition and goals of adopting DevOps, organizational structure, and the matter that customers may not want DevOps. Clear definition is needed for DevOps, because without it people might have different understanding of what DevOps means. People might also have different understanding of the goals regarding DevOps and how to achieve these goals. In Smeds et al.'s (2015) study the interviewees mentioned having a common understanding about the goals and agreeing how to achieve the goal are essential contributors for a successful adoption of DevOps.

Organization's structure can affect DevOps adoption both negatively and positively. In Smeds et al.'s (2015) study, the way an organization is structured was mentioned, for example when discussing communication, common goals and practices, decision making, and systems thinking within the organization. These topics are closely related to several of the capabilities and the cultural enablers (Smeds et al., 2015).

DevOps might not be suitable for customers who require processes and practices including long testing periods or strict deployment procedures. For customer, that might be a reason not to want DevOps. Such processes and practices might in turn not be compatible with the processes and practices of DevOps. DevOps must be implemented in a manner that is compatible with the organizations' processes and practices. (Smeds et al., 2015)

3.4.2 Impediments Affecting Cultural Enablers

Regarding to Smeds et al. (2015) impediments affecting cultural enablers are: geographical distribution, buzzword tiredness, and the assumption that DevOps is more work for developers. In addition, DevOps requires both development and operations skills and knowledge and there might be lack of interest in the "other side".

Smeds et al. (2015) and Walls (2013) agree that geographically distributed operations and development work can create challenges. Regarding to Walls (2013), best change for successful communication is when people sit together or have a common physical space that allows for chance interactions and conversations. Smeds et al. (2015) mentions that communication cannot be done in person and reaching people might be difficult due to different time zones. Social relationships and the environment are fundamental aspects of organizational culture (Smeds et al., 2015). Therefore, the cultural enablers can be hindered by geographical distribution. For teams not permanently co-located, Walls (2013) recommends to get together as long as possible at the beginning of the project. In addition, geographical distribution might also pose other, for example process related, challenges. (Smeds et al., 2015)

People might perceive DevOps as a buzzword. In Smeds et al.'s (2015) interviews, a majority of respondents mentioned the ambiguity of the term or something else regarding how the term is used when asked about how familiar DevOps is as a concept or how to define it. In certain answers, the lack of trust in DevOps as a concept was notable. People might see DevOps as a buzzword for old concepts like continuous integration and test automation. Regarding to Smeds et al. (2015) the mentioned observations suggest that even if people perceive at least a part of the aspects of DevOps as positive, the perception of DevOps as a concept is not always positive. A negative perception might lead to a mindset of resisting change (Smeds et al., 2015).

Some interviewees in Smeds et al.'s (2015) study were concerned that developers become overburdened by extra responsibilities related to operations. The reason behind this fear was a perception that the workload of the developers might increase as the company adopts DevOps. With operations responsibilities, the effort dedicated to pure development work would decrease, unless development resources are added. Another perceived concern was that added operations responsibilities might affect the capacity to focus and work productively. These concerns can result in unwillingness to get involved in

new collaborations and collective ownership. Hence, the concerns might create a mindset that can act primarily as an impediment to the cultural enablers. (Smeds et al., 2015)

In Smeds et al.'s (2015) study, some of the interviewees expressed a concern that DevOps requires development to have in-depth operations skills and knowledge and vice versa. It was argued that people are not able to handle efficiently both development and operations as the areas differ so much in terms of skills and knowledge. The opinion that in-depth knowledge of both areas is needed and that it is better to focus on a narrower area of expertise can create a mindset where people are not open for the cultural traits of sharing, communicating, and collaborating.

Smeds et al. (2015) mentions that there were also concerns regarding the developers' interest in operations work and vice versa. The reason for lacking interest in the other type of work was explained being a result from the nature of experts, who are interested only in their own area. It was also speculated that people might think that they do not fully belong to group, neither development nor operations, when doing DevOps.

3.4.3 Impediments Affecting Technological Enablers

Smeds et al. (2015) mention following impediments affecting technological enablers: monolithic architecture, development and testing environments do not reflect production environments, and multiple production environments. A monolithic architecture can be a bottleneck to rapid continuous build, test, and deployment. The reason for this is that the architecture of the system is closely coupled with how the system is developed, tested, and deployed for use.

A more modularized architecture allows for upgrading smaller parts of the system independently. Smaller parts also mean shorter wait times for build, test, and deployment results. Transforming the architecture or improving the capability of the continuous deployment system is needed to overcome this impediment. However, this can be particularly challenging if the value of such technical change is not evident. These kind of

improvements are easily postponed, for example, in favor of work on new software features. (Smeds et al., 2015)

Some interviewees in Smeds et al.'s (2015) study perceived differences between development, testing, and production environments as a possible impediment. Software might not be properly validated before deployment to production, if there is difficulty simulating production environment in testing environments. Differences between environments can be problematic for continuous delivery and responsibility sharing. This impediment mainly affects technological enablers. However, as the differences might hinder collaboration and shared ways of working, this is also an impediment for the capabilities and cultural enablers. (Smeds et al., 2015)

In addition to the impediment of differences between production and testing environments, interviewed people perceive multiple production environments and differences between them as a possible impediment to continuous delivery (Smeds et al., 2015). Different needs of environments cause complexity. With complexity, automating and having common tools and processes becomes challenging. Even different access rights can cause issues, because when fixing production problems it is essential to have free enough access. The main difficulties that multiple production environments cause are related to deployments and configurations. This is mainly impediment to technological enablers. (Smeds et al., 2015)

3.5 DevOps Maturity Model

The most common and known maturity model in a field of software is The Capability Maturity Model (CMM). Paulk et al. (1993) defines it as a framework that describes the key elements of an effective software process. It also describes an evolutionary improvement path from an ad hoc, immature process to mature, disciplined process. DevOps adoption can be also evaluated using maturity models. In this section, two multilevel DevOps

maturity models are introduced and compared. DevOps maturity models help organization to assess its current situation in DevOps adoption. These maturity models can also be used for the systematic development of organization's DevOps practices.

Figure 3.1 shows Eficode's (2015) maturity model, which specifies five aspects for DevOps, and four levels for these aspects. These five aspects are: organization and culture, environments and release, builds and continuous integration, quality assurance, and visibility and reporting. At the first level, organization does not exercise DevOps practices. Respectively, organization's DevOps practices are in an ideal state at the fourth level. At the second and third level, organization has started to implement some DevOps practices, but they are in their early stages and there are room for improvement.

Figure 3.2 shows Mohamed's (2015) maturity model, which has four aspects for DevOps. Mohamed (2015) defines these aspects as layers. These four layers are Quality, Automation, Collaboration, and Governance. Each layer has five levels of maturity. Mohamed's (2015) maturity model is based on CMMI maturity model (Capability Maturity Model Integration) with five levels of maturity. These levels from the least matured to the most matured are: initial, managed, defined, measured, and optimized.

The two DevOps maturity models presented above have defined similar aspects for DevOps. Mohamed's (2015) model has Quality whereas Eficode's (2015) model has Quality Assurance. Likewise Mohamed's (2015) model has Automation, Eficode's (2015) model has Environments and Release, and Builds and Continuous Integration, which both promote high level of automation. Mohamed's (2015) Collaboration aspect presents somewhat same ideas than Eficode's (2015) Organization and Culture, but the latter also has some things similar than Mohamed's (2015) Governance. Mohamed's (2015) Governance aspect have similar ideas than Eficode's (2015) Visibility and Reporting. Alternatively, the latter have some common things than Mohamed's (2015) Governance, like collecting metrics to improve the whole development process. In summary, these two models present alike ideas with somewhat similar divisions into DevOps aspects, but at the same time they

Figure 3.1: Eficode's (2015) DevOps Maturity Model. Colors are inverted

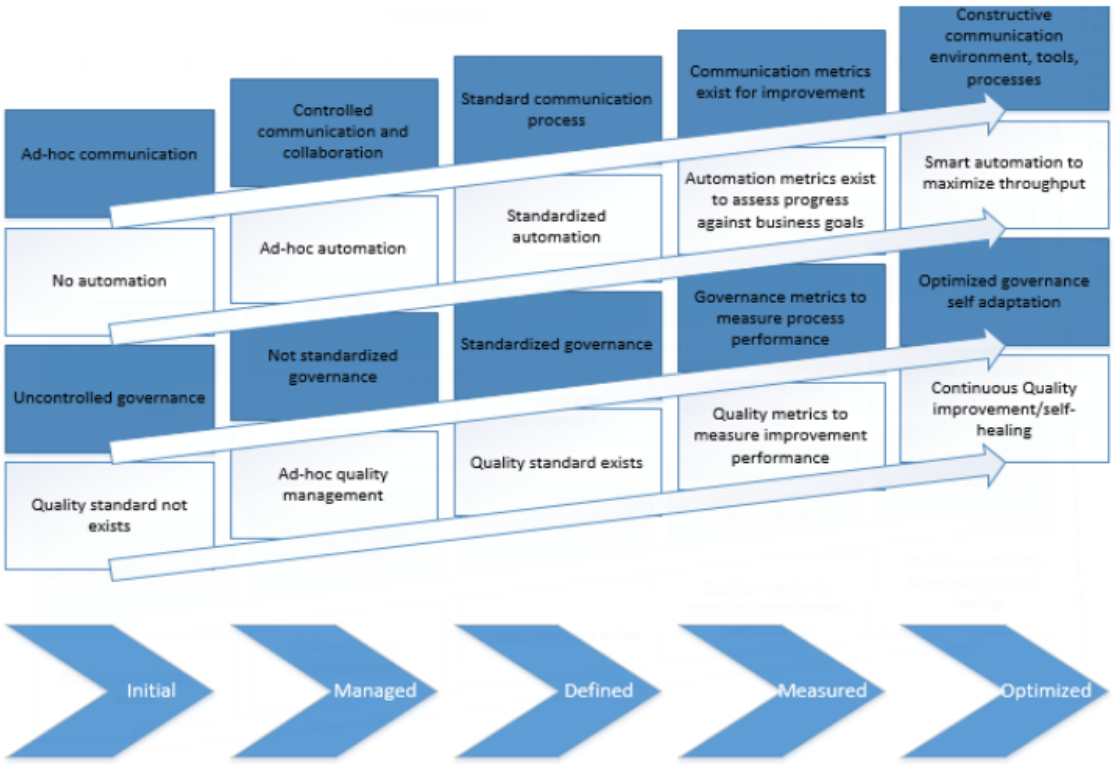

Figure 3.2: Mohamed's (2015) DevOps Maturity Model.

have minor differences.

When comparing DevOps adoption maturity levels presented by Eficode's (2015) and Mohamed's (2015), similarities can be found. At the initial level of both models, there are no automation. Eficode (2015) defines that communication is primarily in writing, whereas Mohamed (2015) defines communication to be ad-hoc. Eficode's (2015) model specifies that at the initial level, quality assurance is conducted completely by hand and primarily after development. Mohamed's (2015) states that quality standards does not exist at the initial level. Regarding Mohamed (2015), governance is uncontrolled at this level. This can be seen in relation to Eficode's (2015) definition that design, development and quality assurance are separate from each other. This is because Mohamed (2015) defines the governance layer to be primarily responsible to control how different layers work seamlessly together.

The next level of both maturity models states that there are some kind of automation

activities. Eficode (2015) defines that some releases are automated and Mohamed (2015) states automation is ad-hoc. Regarding to Eficode (2015) work is conducted in teams at this level, whereas Mohamed (2015) define that communication and collaboration is controlled. In Eficode's (2015) model, there are some quality assurance efforts like unit testing and static code analysis. Respectively, Mohamed (2015) defines that ad-hoc quality management exists at this level.

The third level of both models presents a state where all DevOps aspects presented in the models are in place and working relatively efficiently. Mohamed's (2015) model has one level more than Eficode's (2015) model. Last two levels in Mohamed's (2015) model are named Measured and Optimized and they present roughly the same ideas as the last level in Eficode's (2015) model. The last level in Eficode's (2015) model highlights using metrics for continuous improvement to achieve even a better state of efficiency and quality. To summarize, both of the models present almost the same ideas, but from different viewpoints. They agree which aspects are important when organization is adopting DevOps practices.

3.6 Chapter Summary

This chapter covered organization's DevOps adoption. Smeds et al. (2015) have studied capabilities, enablers, and impediments which will affect DevOps adoption positively or negatively. Capabilities are processes that organization should be able to carry out. Enablers can be divided into two categories: cultural enablers and technological enablers. Cultural enablers are traits that DevOps team and organization should exhibit. Technological enablers mainly emphasize the need for automating tasks. Impediments can complicate DevOps adoption from the perspective of capabilities, cultural enablers, and technological enablers. In some cases, an impediment can negatively affect several subclasses of capabilities and enablers. Table 3.1 presents main subclasses of capabilities, enablers, and

Capabilities	Continuous operations Feedback Ability to recover from failures
Cultural Enablers	Open and effortless communication Shared responsibilities, common goals, and extensive collaboration Supportive working environment Executive sponsorship and technical leadership
Technological Enablers	Task automation
Impediments Affecting Capabilites	Unclear definition and goals of adopting DevOps Organizational structure Customers may not want DevOps
Impediments Affecting Cultural Enablers	Geographical distribution Buzzword tiredness Assumption that DevOps is more work developers
Impediments Affecting Technological Enablers	Monolithic architecture Development and testing environments do not reflect production environments Multiple production environments

Table 3.1: Main subclasses of DevOps adoption capabilities, enablers, and impediments

impediments according to Smeds et al. (2015) and Walls (2013).

DevOps maturity models can be used for simplified assessment of an organization's DevOps maturity. This chapter presented Eficode's (2015) and Mohamed's (2015) maturity models. In addition, the models were compared with each other. Maturity models are suitable for identifying problems in organization's DevOps adoption. After certain problem or an impediment is identified, a more detailed analysis is needed for finding the root cause

for the problem. This is because the maturity models present the whole concept of DevOps in a simplified manner and organizational problems are usually complex.

4 Case: Introduction and Research Methods

4.1 Introduction

In this chapter, the case study part for this thesis is introduced and defined. The purpose and objectives for the case study are defined and set. In addition, the brief introductions of the system studied, the organization using the system, and the environment where the organization operates are presented. Finally, research methods used in the case study are presented.

4.2 Defining the Case Study

4.2.1 Purpose and Objectives

Purpose of this case study is to examine status of organization's system development from a DevOps point of view. The organization in question is a large telecommunications company and the system studied is a complex system spanning across business functions like order management, delivery, and network provisioning. The case study was made because the organization was interested if DevOps methods and concepts accelerate its system development and decrease problem situations related to deployments and system maintenance. The particular system was selected, because it had most interest for the

company. The results might had been totally different if a different system would have been selected.

The case study differs from most of the earlier DevOps studies, because the company is not doing the technical system development itself, but the development work is done by a vendor. Technical in this context means actual programming and implementation work. However, the company can still propose the best DevOps practices for the vendor, thus improving the efficiency and quality of the system development and the system itself. Nevertheless, at the same time this means that in-depth assessment of technical details related to the system development could not be done, because there is no visibility to these details.

Objective is to create a high-level assessment related to the DevOps maturity of the organization and the system. Furthermore, the objective is to identify some clear problem points regarding DevOps adoption. This DevOps maturity assessment can then be used as the initial starting point in organization's journey to adopt DevOps in its development and operations practices. The hypothesis is that DevOps maturity is likely on a low level. This is because the system is very large and complex, and the organization structure used in system development is not ideal for DevOps practices. Also, it is probable that some of the findings will be more organizational problems than technical problems. This would be expected because adopting DevOps is always also an organizational change, like mentioned earlier in this thesis.

Research questions for this thesis are presented below. RQ1 has been mainly answered in Chapter 2. Chapter 3 provides a literature context for RQ2. After that, Chapter 5 continues answering RQ2 on the basis of case study.

RQ1 What is DevOps and what does the term contain?

RQ2 Which factors should an organization consider when adopting DevOps?

4.2.2 Organization

The company where this case study was conducted has several thousands employees and decades long history. The company is a buyer and has bought the core system studied from a system vendor. The organization is referred to from now on as the 'client' or 'company'. The vendor is foreign and the development work is done off-site. In addition to this core system, there are also customer channels systems, like self-service portals, related to this core system. These systems are developed by another near-shore vendor. These systems are then connected to the integration platform used in the company by third vendor. To summarize, the system is developed in a multivendor environment. This case study is emphasized on the core system development, because this kind of concentration fits better to the scope of a master's thesis.

To simplify, the requirements analysis and the design from business perspectives are done by the client and technical design and implementation is then done by the vendor. System maintenance is done by the client and it owns the servers and infrastructure used to run the system. The vendor can utilize these servers in different phases of the development work. In addition, the vendor is responsible for installing the system in the test environments. The client is responsible for installing the system in the production and maintenance environments.

During the period of a year, the company adopted SAFe (Scaled Agile guide, 2016b). SAFe is an abbreviation for Scaled Agile Framework. To simplify, it is a framework to scale agile development into the enterprise level incorporating multiple Scrum teams. SAFe was adopted to provide efficiency, quality, and transparency to the development process. In addition, it was selected because the framework provides a strict control which is uncharacteristic for agile methods and fundamentals. This is beneficial, because the development resources are off-site.

SAFe has three levels: portfolio, program, and team. These levels are not defined in this thesis due to the limited scope of the case study. For the same reason, SAFe is not

further presented. However, there is a component named DevOps in SAFe. This is briefly analyzed in the next chapter.

4.2.3 Environment

The system is central part of the large business transformation process. The main goals of this transformation process are simplifying processes and making them more efficient. A simple and efficient process would then probably be more cost-efficient. Currently, there are lots of overlapping in the processes and almost similar products might be delivered from different systems. Also, some processes contain redundant steps.

It is objective that the transactions with business-to-business customers and business-to-consumers customers can utilize the same processes, information system processes, and system environments. In addition to cost-efficiency, simplified overall processes would provide faster reactivity, thus new products and campaigns could be launched faster.

4.2.4 System

In telecommunications industry, systems can be divided into Business Support Systems (BSS) and Operations Support System (OSS). BSS includes finance, billing, decision support, customer relationship management (CRM), contracting, and customer self-care systems. OSS includes provisioning, order management, ticket management, and service assurance systems. (Pollet et al., 2006). The system examined in this case study covers both BSS and OSS functionalities with slight BSS emphasis. Handling network infrastructure is out of scope for this system. The system consists of several components, each having different functionalities and responsibilities.

To summarize and simplify, the system provides a capability to configure telecommunications product offerings and to offer products from these offerings to the customers. The system has functionalities for all of the functions in that process chain. There are components for the following functions: customer information management, sales and orders,

billing, offering configuration, and network provisioning. In addition to these internally used components, there is also channels layer which provides self-service and sales tools to external end customers. At first, the system was only used with business-to-business (B2B) customers, but there is a plan to migrate some of the business-to-consumer (B2C) customers to the system in the near future.

The system has been in development for about five years and the development work is done by external vendors. It was bought as a commercial off-the-self product initially, but there have been modifications and customizations during the years. The system has hundreds of internal users in the company. Also, some customer facing components like online self-service tools are in development, so the system will have also external users in the future.

As mentioned in the earlier section, the client is responsible for installing the system in the production and maintenance environments. The vendor will provide a built binary package, and then the client will install the package in its servers. This means that the client has no real visibility to the internal aspects of the system like code quality or coding conventions used in the system.

4.3 Research Methods

In order to answer research question, qualitative research methods were selected. This study's research approach follows a single case study design principles. The study itself is of exploratory, i.e., that the aim is to discover new hypotheses and explanations rather than to verify the existing theories. 14 interviews were conducted with the key personnel in the project for assessing the DevOps maturity of the system. The interviews were conducted during a four-month time period from January to April 2016. Lengths of the interviews varied from 12 minutes to 50 minutes, 27 minutes being an average length for an interview. Some of these personnel were the company's own employees and some of them

Interviewee	**Title in the organization**
Interviewee_1	Test Consultant
Interviewee_2	Test Manager
Interviewee_3	SAFe Trainer
Interviewee_4	Program Lead
Interviewee_5	Release Manager
Interviewee_6	Test Consultant
Interviewee_7	Operations Manager
Interviewee_8	Finland CIO
Interviewee_9	System Upgrade Manager
Interviewee_{10}	Head of Delivery and Supply Management in IT Finland
Interviewee_{11}	SAFE Consultant
Interviewee_{12}	Manager in Small Development
Interviewee_{13}	Head of Testing
Interviewee_{14}	Group Manager in IT

Table 4.1: List of interviewees and their title

were external consultants working in the project. Titles of the interviewees varied from configuration manager to a product owner, so some of them were more technically oriented and some of them were more business oriented employees. The company and these employees wish to remain anonymous for this case study. Table 4.1 presents interviewees and their titles in the organization.

Semi-structured open-ended questions were used in the interviews. The interviews were started by asking the same questions from every interviewee. These questions were related to the interviewees work responsibilities and initial perception of DevOps and attitude towards DevOps. After these initial questions, the questions were customized according to interviewees' work responsibilities. Work responsibility information acquired from initial questions was then used to ask follow-up questions in some interviews. In the majority of the interviews, a transcription was created after interview.

Data from the interviews was compared with DevOps capabilities, enablers, and

impediments found in Smeds et al. (2015) and Walls (2013) studies to identify if there were similarities. Also, DevOps capabilities, enablers, and impediments unique for this project were analyzed. Lastly, Eficode's (2015) DevOps maturity model was used to determine DevOps maturity of the project according to findings from the interviews.

5 Results – Assessing DevOps Adoption

5.1 Introduction

The main objective for the case study is to assess DevOps adoption in the particular information system project. Background for the studied organization, environment, and system are provided in the earlier chapter. It also defines the research methods used in this case study. DevOps adoption capabilities, enablers, and impediments defined by Smeds et al. (2015) and Walls (2013) are used to structure this chapter. Findings from the interviews are set on these categories and compared with the findings in Smeds et al. (2015) and Walls (2013) studies. Section titles are after these categories. If adoption detail which is not addressed in Smeds et al.'s (2015) and Walls (2013) studies is found in the interviews, it is set to the one of the earlier mentioned categories, when possible. If the found adoption detail does not fit these categories, it is presented in Section 5.8. The key findings of each aspects are presented in Table 5.1. Lastly, the DevOps maturity of the project is assessed using Eficode's (2015) maturity model.

5.2 Capabilities

5.2.1 Continuous Operations

The term continuous is defined in Chapter 3 as in small increments without delay. Continuous deployment is mentioned being an example of a capability like this. The organization

Subsection Title	**Key Finding**
Continuous Operations	Overall technical quality of the system is low
Ability to Recover From Failures	Lack of recovery testing and improvement of current monitoring solution
Open and Effortless Communication	Difficulties in internal and external communication
Shared Responsibilities, Common Goals, and Extensive Collaboration	Lack of common development methods and standards. Increasing technical debt
Supportive Working Environment	Geographical distance hinders trust building. Finding balance between trust and control
Executive Sponsorship and Technical Leadership	Managers should have patience with the transformation project
Task Automation	There is an automated deployment process and some automated tests. Vendor has problems in version control
Organization Structure	Vendor has hierarchical organization culture
Customers May Not Want DevOps	Customer process regulations might hinder DevOps adoption
Geographical Distribution	Communication is more challenging with off-site vendor
Buzzword Tiredness	Personnel have somewhat positive outlook on DevOps
Monolithic Architecture	There are problems in system architecture
Development and Testing Environments Do Not Reflect Production Environments	Configuration differ between testing and production environments
SAFe Adoption	Beginning of SAFe adoption has been positive
System Team and DevOps in SAFe	Having dedicated system team has had positive impact
Vendor Relationship	Vendor relationship is contradictory
Issues in Testing	Majority of the testing takes place at the end phases of development
Challenges in Project Management	Progress of the project is difficult to estimate

Table 5.1: List of key findings per subsection

should have an automated testing and deployment pipeline and streamlined collaboration between developers and operations. In this case study, the studied organization is rather far from this kind of tool chain regarding the studied project. However, the company has started to invest in automation and there has been progress in automatic deployment and test automation projects.

Several interviewees mention that the overall technical quality of the system is low. An interviewee emphasises this by commenting that the company is very far away from DevOps model where changes are deployed directly to production. He explains this by saying that the software development or quality assurance of the vendor is not mature enough for exercising that kind of DevOps model.

Despite these difficulties mentioned above, the company has made progress regarding automation. It has implemented automated deployment procedures in collaboration with the vendor. These automated procedures have already decreased errors in deploying new versions of the software. In addition, the company has automated some of acceptance tests it does after it gets the new system version from the vendor. To summarize, the company is moving in a right direction regarding continuous operations. Also, management has shown its support for increasing level of automation and quality assurance in software development processes regarding the system.

5.2.2 Feedback

Like mentioned in Chapter 3, data collected from system monitoring is used to improve and optimize the service. Currently, IT operations of the company is doing monitoring using Nagios. However, current monitoring is not extensive enough. For example, alarms are defective. This is why the company has started a monitoring side project. The plan is to create more extensive monitoring.

5.2.3 Ability to Recover from Failures

As stated in Chapter 3, a DevOps organization should be able to recover from service failures caused by the service infrastructure or software defects. This recovery should happen without a delay. A monitoring infrastructure should detect failures immediately. The company has started to improve the monitoring solution, so there is progress regarding detecting failures. An interviewee mentions that recovery testing has not got much attention, because the working hours of the testing team goes to bug reporting and defect management of the core system. In addition to detection, the company should have a contingency plan for reacting to the failures. Lack of recovery testing might hinder the contingency plan. There is a plan to create high availability environments for the system until the end of the year. This kind environment will make the system more fault tolerant.

5.3 Cultural Enablers

5.3.1 Open and Effortless Communication

Open and effortless communication is one of the cultural enablers mentioned in Chapter 3. Regarding to Walls (2013), in organization culture incorporating open and effortless communication, every team member is able to ask questions from other team members and expect to get an answer or help in finding one. Some interviewees mention that there is mismatch in communication between the managerial layer and personnel who are doing more hands-on work with the system. There is slight lack of realism on the manager level, which sees matter in a more positive manner compared with people doing the hands-on work. An interviewee criticizes the vendor that sometimes it does not have courage to admit that schedules of the client are too ambitious. Another interviewee adds that the vendor sugar-coats matters and communicates on a different level with managers than with personnel who are doing the hands-on work.

Communication is done in English, which is not the mother tongue for either the vendor

or the client. This sometimes creates natural challenges in communication. An interviewee mentions that common concepts have regularized during the years of collaboration, which have made communication easier. Another interviewee mentions that there have been situations where the vendor has a problem to which the client would had had an answer. Still, the vendor did not come forward with the problem. On the contrary, there have been situations in which the vendor exaggerates the importance of a minor problem. The interviewee adds that in his opinion these kind of things happen because the right level of communication and methods of collaboration are still sought.

When an interviewee is asked if the reason for vendor's sugar-coating is that the client wants too tight schedules, he answers that in his opinion that is not the case. He emphasizes that the client has communicated that the Scrum teams of the vendors should give honest estimations about how much work they can do in a certain period of time. The client concentrates on understanding the system as a whole, what needs to be done, and prioritizing the work. After that, amount of work a Scrum team takes depends on their self-evaluated capacity.

One of the interviewees mentioned that a special characteristic in communication has been so called corridor rumours at the client office. For example, if someone tells that performance of the system is not at good level, this comment will be truth soon. This will raise suspicion among the employees. Monitoring data supports and fosters open communication. In addition, this data should be prominently displayed and available for everyone. This kind of monitoring data could reduce rumours, because everyone can see the performance of the system from the collected metrics. Another interviewee mentioned that a monitoring side project regarding the system has been started in the company.

5.3.2 Shared Responsibilities, Common Goals, and Extensive Collaboration

Shared responsibilities, common goals, and extensive collaboration are cultural enablers which will positively contribute to DevOps adoption. Regarding to Walls (2013) these enablers can be promoted by setting a uniform incentive for the team. However, in this case study the vendor is doing the most of the technical development work. Therefore, setting a uniform incentive across the vendor and the client might be difficult. One of the interviewees criticized the contract between the vendor and the client. In his opinion, the contract does not obligate the vendor to be responsible for the quality aspects of the system. Problems of the contract are further analyzed later in this chapter.

Related to the matter stated in the previous paragraph, an interviewee mentions that if the vendor says they will improve their actions, there is no concrete evidence to support that claim. In this context, concrete evidence could be assigning more personnel to the project, personnel trainings, or creating code review processes to the development. The client is creating a guideline document regarding agile development, testing, and quality assurance. After the document is created, the vendors which the client is doing development with should comply with the guidelines in the document. This kind of explicit document might help to set the development standards and create a basis for the shared responsibilities, common goals, and extensive collaboration.

An interviewee commented that system development has been rather business-oriented and there is always a commercial pressure to publish. For example, throughout the course of the project, resources have not been dedicated to defining and developing non-functional requirements. In addition, testing framework or mocks have not been built. Because of these matters, technical debt increases all the time, according to the interviewee. Recently, there has been more focus on the non-functional requirements and the client has realized that the there are lots of things in the system that have been implemented with low quality. Fixing these issues will be a lengthy process. This kind of contradiction between

commercial pressures and technical maturity might hinder working towards a common goal and extensive collaboration.

5.3.3 Supportive Working Environment

Walls (2013) mentions respect and trust regarding supportive working environment. Regarding to Smeds et al. (2015), supportive working environment is a cultural enabler for DevOps adoption. Trust is an essential component of achieving a DevOps culture. Geographical distance between the client and the vendor has probably hindered building trust between these two parties. In addition, communication challenges and false promises mentioned in this chapter earlier, have presumably impeded trust building. The whole DevOps culture builds upon trust, so this is an issue which should be improved somehow.

When asked about risks in building DevOps competence, an interviewee in a high managerial position acknowledges that the company needs to find balance between trust and control in its working methods. DevOps or agile working methods needs trust and freedom for being effective. He adds that it needs to be carefully considered how the right balance is found between trust and control, so that the trust in not failed. In his opinion, the quality thinking needs to be integrated into the team level.

5.3.4 Executive Sponsorship and Technical Leadership

There is a need for executive sponsorship and technical leadership in DevOps adoption. An interviewee in a high managerial position mentions that in his opinion executive sponsorship is at good level regarding agile transformation in the company. He adds that slight impatience is present and the pressures are high, but he has a positive outlook regarding the transformation. Another interviewee mentions that he hopes that the management will have patience regarding the transformation. He comments that the organization is in the most challenging phase of the learning curve where the organization is seeking best practices.

The company has certain employees in the leading role of the mentioned SAFe adoption. If the company wants to invest in DevOps adoption, maybe it could set some employees in DevOps lead roles. Like mentioned earlier, the role of technical lead is a combination of evangelist, tools expert, and a process subject-matter expert. Technical lead can be a person or a team of people. These employees would then be internal guides in DevOps adoption.

5.4 Technological Enablers

5.4.1 Task Automation

Technological enablers emphasize a need for task automation. Following concepts and tools are examples of technological enablers: build automation, test automation, infrastructure automation, and configuration management. Various findings related to task automation were found in the interviews. Some of the findings were related to recent progress in automation, but many underlined the need for improving a technological foundation of the system and processes before end-to-end automation can be implemented.

There has been an effort to automate deployment processes. First automatic deployment to production was done in February. To make automatic deployments possible, significant architectural changes has been made to the system. An interviewee mentioned that more automated installations have improved the reliability of installations. This has saved a lot of time. However, there are still some limitations in the automation process and technological foundation of the system. These limitations are analyzed later on in this section.

Another interviewee mentions that there are good plans for automation. He adds that the company knows that it has certain constraints related to methods and technology which might hinder implementing DevOps practices at ideological level. He underlines that it is extremely important that the company has understood the importance of automation and conventions related to it. The path to successful DevOps adoption is unique to each

organization. Therefore, it is beneficial that the company has identified certain constraints which might affect its DevOps adoption.

Despite the mentioned progress in automation, there are still some limitations. Such limitations are: ad-hoc configuration management, only some of the components can be automatically built nightly, and lack of proper source code version control. Configurations are done by hand depending on the package which the vendor delivers. Configuration files are stored to the version control. An interviewee mentions that there have been problems related to configurations in testing environments when testing is moved to a new environment. These problems are mainly related to integrations. So, there is a sign that configuration management could be improved. Configuration management tools and Infrastructure as Code concepts could solve some of these problems.

The vendor has an automatic nightly build system which builds new component packages every night. However, not every component is in the nightly build system. Hence, there is still room for improvement. Nightly builds enable triggering, which makes possible to build new packages when they are needed, for example at daytime.

The current build pipeline includes version control only for binary packages, because the vendor has intellectual property rights (IPR) to source codes. Hereby, the client has very limited visibility to the version control conventions of the source code. An interviewee mentions that the vendor has showed Powerpoint presentations about its version control conventions, but the reality does not match these presentations. Sometimes there has been version mismatch between components or broken component packages. After these kind of problems, nothing usually works. Regarding to the interviewee: “Version control usage is still in its infancy.”

Another interviewee adds that regarding the latest information, the vendor uses seven different version control systems. A binary package is compiled from these different version control systems, which is then handed to the client. The interviewee adds that this sets a challenge, because handling binary versions are more difficult than handling

source code versions. The vendor should standardize its source code version control and implement some best practices, because version control is a base layer in the automatic build pipeline.

The company has automated some of the acceptance tests. In addition, automation is used for smoke tests, which will test integrations. The automated tests are implemented with Robot Framework. The automated regression test set has been increased gradually on the B2B customer side. Currently, the test set decently tests basic functions. There is a plan to implement automated tests into the B2C customer side when the next release will be activated and it is more stable. The focus of test automation is on making overall time spent on testing shorter.

An interviewee mentions that automation has its challenges, because checks need to be done from various systems, which do not have web interfaces. Another challenge with automation is that the user interfaces frequently changes. A lot of time and different needs to be put to the maintenance work. Regarding the interviewee: “Once the script is implemented and it is functional, the next release of the system might break it”.

The interviewees were not confident that the vendor is doing unit testing in the development. An interviewee mentions that the vendor should do unit testing, but in reality they are not doing it. The vendor has implemented some automated tests, but the test set is limited and the tests test old functionalities which have worked from the start. So, these tests do not add value to the quality assurance. Another interviewee adds that after talking with the developers of the vendor, he found out that all of the developers did not know what unit tests mean.

The fact that the company gets a binary package from the vendor creates challenges for the quality assurance. Acceptance testing is always done after a new package is delivered if there are a certain number of findings which need to be fixed. After these issues have been fixed, a new package is delivered and new acceptance tests are executed. Regarding to an interviewee this kind of quality assurance is more close to waterfall-like quality assurance.

However, he adds that it might be the most efficient method in this context where a black box like the binary package is delivered.

In an ideal DevOps approach, automated tests are favoured and tests should be written at unit, integration, and acceptance test levels. Manual testing can be slow, error-prone, inconsistent, and not always fully repeatable. However, according to the comment of an interviewee, optimizing and automating testing is not currently the actual problem in the system. Regarding to him, the problem is the amount of the bugs and issues found in the testing, not the time which test execution takes. Hence, the company should fully invest in test automation after the overall quality of the system is improved.

5.5 Impediments Affecting Capabilities

5.5.1 Unclear Definition and Goals of Adopting DevOps

As stated by Smeds et al. (2015), the unclear definition and goals of adopting DevOps might hinder DevOps capabilities. If DevOps practices are adopted in the project, a clear definition about DevOps should be made in the context of the project. Defining the term and practices will decrease miscommunication and wrongly aligned expectations.

5.5.2 Organization Structure

Smeds et al. (2015) mention that organization structure can affect DevOps adoption both negatively and positively. Some of the interviewees mentioned that the big size of the project introduces natural problems for project management. Related to that, an interviewee mentioned that adopting SAFe has eased the mentioned problem, because it has clear roles for employees. With clear roles, employees will know more easily what is expected of them and what are their responsibilities. However, SAFe adoption is still in progress.

An interviewee mentioned that one problem is that vendor has rather hierarchical organization culture. Making a straight contact with developers has been difficult, because

there is a managerial layer between developers and the client. Sometimes this hinders the progress of development. Another interviewee adds that getting in contact with correct team members is difficult. Instead, someone who speaks on behalf of the team is reached. Recently, there has been improvement regarding that problem with the Scrum of Scrums convention where the Scrum masters of the vendor report directly to the person who is in charge of the development on the client's side.

A couple of interviewees mentioned that vendor should have more cross-skilled teams. Now the developers are specialized in a certain component or certain configuration. There have been problems because the developer implementing a change to some components does not understand parallel effects the change causes in other components. Also, lack of cross-skilled developers has sometimes slowed development work.

5.5.3 Customers May Not Want DevOps

This subsection of DevOps adoption is not relevant to this case study. This is because the case study is made from the perspective from the customer organization. In this case study, the customer is driving the DevOps adoption instead of the vendor, so there are no problems related to the fact customer would not want DevOps. However, strict processes on the customer side might hinder DevOps adoption.

5.6 Impediments Affecting Cultural Enablers

5.6.1 Geographical Distribution

The cultural enablers can be hindered by geographical distribution. Some of the interviewees mention that having an off-site vendor has been a challenge in the project. Communication is more challenging when the developers are not in common physical space with the employees of the client.

As mentioned earlier, Walls (2013) recommends for teams not permanently collocated

to get together as longs as possible at the beginning of the project. The client tries to mitigate this impediment created by geographical distribution by travelling to the vendor. These trips are usually related to SAFe ceremonies like the release planning event. One of the interviewees mentioned that he has perceived that creating requirements specifications has been more efficient when the vendor has sat on the same table.

5.6.2 Buzzword Tiredness

In Smeds et al.'s (2015) study one found impediment affecting cultural enablers was that people might perceive DevOps as a buzzword. In the interviews conducted for this research, the majority of the interviewees had a positive attitude towards DevOps. None of the interviewees referred to DevOps as buzzword although one of the interviewees mentioned that in his former projects practices similar to DevOps were exercised, but they were not called DevOps at the time. One interviewee had had bad past experiences about bigger projects which were done with full agile methods.

Some of the interviewees mentioned that in their opinion, the suitability of DevOps is always a project and context related. One interviewee mentioned that DevOps might make deployments more efficient and reduce defects in the process. Some of the interviewees admitted that they do not have in-depth knowledge about DevOps and can only comment based on their current knowledge, which might be defective. Creating a clear company-wide definition of DevOps might reduce misunderstandings during the DevOps adoption.

5.6.3 Assumption that DevOps is More Work for Developers

Regarding to Smeds et al. (2015), developers might assume that DevOps is more work for them and this impediment would affect cultural enablers. This matter can not be examined based on this case study, because the developers of the vendor were not interviewed. In addition, it could be assumed that the client wants its own people to run operations. However, DevOps adoption might require developers to learn new tools and processes,

even if the employees of the client have the responsibility of the operations.

5.7 Impediments Affecting Technological Enablers

5.7.1 Monolithic Architecture

Monolithic architecture can be a bottleneck to rapid continuous build, test, and deployment. The main components of the system studied can be build separately. The components communicate with each other, so the communication channels need to be compatible. Otherwise the build result will be a broken package.

There were comments on the interviews which suggest that the system is not properly decoupled at the lower levels. For example, one of the interviewees mentioned that sometimes a certain part of the system is broken, even if there have been no changes to that part. Also, bugs with the same subjects need to be created. One of the problems is also that configuration changes need to be made to several locations manually. These kind of problems are likely to be an impediment to task automation.

Two of the interviewees mentioned that running the system requires extensive hardware resources compared with the fact that the system is relatively simple. According to an interviewee, the solution to this problem should be optimizing the system, not buying more hardware. Another interviewee commented that there are lots of problems in the system architecture. This is not related to monolithic architecture per se, but it is a sign of bad architecture. Bad architecture might hinder exercising rapid continuous build, test, and deployment processes.

As mentioned earlier, in cases where architecture transformation is needed, the change might be challenging to reason if the value of such technical change is not evident. Hence, new software features might be favored over this kind of change. Related to that, an interviewee commented that if the quality problems in the system are related to bad architectural decisions or a badly designed database, the common development work

should be stopped and the architecture should be fixed. He mentions that this kind of change is difficult to sell to the management. Nevertheless, if that is not done, there will always be technical debt.

5.7.2 Development and Testing Environments Do Not Reflect Production Environments

Another impediment affecting technological enablers mentioned in Smeds et al.'s (2015) study is that development and testing environments do not reflect production environments. In the system studied, development and testing environments are similar to production environment with minor exceptions. In testing and development environments, structural tests similar to production can be executed.

The system environment has multiple servers. There are similar servers in development and testing environments than in production environment. As mentioned, there are minor exceptions between development, testing, and production. Virtual servers are used in development and testing environment, whereas bare-metal servers are used in production. So, there are differences in the hardware used. In addition to differences in the hardware, configurations somewhat differ. These kind of small differences between testing and production environments might pose a problem for continuous delivery, because software might not be properly validated before deployment to production.

5.7.3 Multiple Production Environments

As mentioned in the Smeds et al.'s (2015) study, multiple production environments can be a problem for continuous delivery. The system studied has only one production environment, so this impediment is not relevant to this case study. However, this impediment needs to be considered if there will be more than one production environment in the future.

5.8 Other Findings Relevant for DevOps Adoption

5.8.1 SAFe Adoption

Among the majority of the interviewees, SAFe has had a rather positive welcome. Interviewees mention that the adoption is still in progress and people are still learning common terminology and methods. An interviewee mention that it is more difficult to integrate the vendor into the SAFe way of working than to motivate their own personnel to adopt SAFe. He adds that the adoption will be a rather long process, but the start has been positive.

Another interviewee comments that in his opinion SAFe is meant to scale already agile organization. He adds that agility should be worked out before implementing SAFe. In addition, another interviewee mentions that he has a sceptical outlook of using agile methods in large projects. In his opinion, the basic design should be done before the implementation work is started. He adds that if design, implementation, and testing should happen with one sprint, there usually is not much finished work at the end of the sprint.

Despite SAFe adoption is not directly related to DevOps adoption, it is worth mentioning in this case study, because it is major change in working methods within the project. Additionally, regarding to Scaled Agile guide (2016b), DevOps is part of SAFe. This detail is further analyzed in the next subsection.

5.8.2 System Team and DevOps in SAFe

A system team is special agile team that is typically chartered to provide assistance in building and using the agile development environment infrastructure, including continuous integration, integrating assets from agile teams, and performing end-to-end solution testing (Scaled Agile guide, 2016c). The company studied has had system team for about six months. The beginning of the team has been rather successful. For example, it has participated in enhancing the performance of a certain component, started a monitoring project, and optimized deployment process. In the future, the goal for the system is to

create practical development landscape and detecting problems early when they are not yet problems.

Like mentioned earlier, having a technical lead is beneficial for DevOps adoption. System team could take that kind of technical lead role, because it has technical knowledge of the system. System team could then recommend best practices and technologies. It could also provide assistance in the adoption of recommended best practices and technologies.

Regarding to Scaled Agile guide (2016a), DevOps part in SAFe means mechanisms for the tighter integration of development and deployment operations. DevOps in SAFe deals more with technical details of continuous integration than organizational matters. To summarize, it mostly describes how to create a deployment pipeline.

An interviewee comments that in his opinion, that is one of the weaknesses of SAFe. He adds that DevOps in SAFe can not be spoken as full-scale DevOps. However, DevOps ideology does not dictate how things are done, only that the chain from development to production is optimized. Despite the fact that DevOps viewpoint in SAFe is technically-oriented, it specifies best practices regarding the creation of a functional build pipeline. For example, Scaled Agile guide (2016a) mentions following practices: build and maintain a production-equivalent staging environment, maintain development and testing environments to better match production, and deploy to staging every iteration; frequently deploy to production.

5.8.3 Vendor Relationship

As mentioned, there have been communication problems with the vendor and problems with the technical maturity of the product that the vendor offers. An interviewee mentioned that difficulty of the project has been a surprise to the client. The client has had to comment on very technical matters. In addition, the client has been a driving force for making the project more agile. Another interviewee comments the vendor has challenges leading the conversation with the client in a way which will produce solution proposals.

However, the cooperation with the vendor has also its positive side. An interviewee comments that the client has influence how the product will be developed and the vendor listens to the wishes of the client. The interviewee mentions that the cooperation can be seen as a strategic partnership.

5.8.4 Issues in Testing

The major problem in testing and quality assurance is that the majority of it takes place at the end phases of the development and that the client has a big responsibility in testing. The vendor is doing testing, but a number of bugs and issues found in the acceptance tests of the client is still high. The high number of bugs makes testing slow and defect management challenging. An interviewee mentions that the ideal situation would be that testing on the client's side would be a cursory testing of the business processes. In this context, the term cursory testing means that the testing can be done from user's perspective and it does not require technical knowledge of the internal functionalities of the system.

Because the testing takes a long time and substantial effort, the emphasis of the testing has been on functional testing so that the system can be deployed on time. Because of this, testing of non-functional requirements, usability, and recovery have often needed to be outscoped.

5.8.5 Challenges in Project Management

Some interviewees comment that there are problems in project management. They mention that it is difficult to estimate how many increments are needed for some feature or functionality to be production-ready. In addition, visibility to the questions like what should be done, what has been done, and in which time is poor. Another problem is that schedule estimations are not on target. An interviewee comments that there have been situations where five sprints are allocated for implementing some feature and during the last sprint it is realized that 30 percent of the content is still unimplemented.

5.8.6 Applicability to Telecommunications Business

One of the interviewee comments that he doubts if DevOps ideology is applicable to the core business of a telecommunications operator. Reason for this is the fact that basic products seldom change. This point is valid if DevOps is seen as a capability to rapidly change products or create new products. Nevertheless, DevOps can be seen as a set of technical and non-technical best practices for software development. These practices will help to create quality software products and organization culture where the foundation is based on constant learning, continuous improvement, and collaboration. In addition, telecommunications operators are increasingly providing services. DevOps practices and ideology could positively contribute to delivery of these services.

5.9 Assessing Maturity Using DevOps Maturity Model

Eficode's (2015) DevOps maturity model is presented in Figure 3.1. Using findings from the interviews, the assessment of the DevOps maturity of the project is done using the maturity model. However, the applicability of the model is questionable in the context of this case study. This is because there is no proper visibility to the development and testing practices the vendor exercises. The maturity model is divided into the following sections: organization and culture, environments and release, builds and continuous integration, quality assurance, and visibility and reporting. There are four maturity levels labeled from 1 to 4, 1 being the least mature level and 4 being the most mature level. Findings are summarized in Table 5.2.

5.9.1 Organization and Culture

Organization and Culture are between levels 1 and 2. Design, development, and quality assurance are separate from each other. The client is doing design at least from business perspective, the vendor does the development, and quality assurance is mostly the

responsibility of the client. However, with the adoption of SAFe, the product owners of the client have a Scrum team from the vendor. The product owner and the Scrum team are geographically distributed, but Scrum ceremonies like daily Scrums are held via teleconferencing solution. So in a sense, work is conducted in teams. However, the separate testing team of the client is primarily responsible for testing.

5.9.2 Environments and Releases

Environments and Releases are on level 2. The system has several components and there is the deployment automation process. However, the configuration is done by hand and only some components can be automatically build nightly. This fact indicates that Environments and Releases could also be on level 1.

5.9.3 Builds and Continuous Integration

Builds and Continuous Integration are on level 1. There is no visibility if product integration is automatic on the vendor side. The configuration is controlled manually, but some deployments are automated. Binary packages are stored to version control, so those can be seen as artifacts. The interviews did not provide insight about the existence of change log management. However, there is no visibility to source code level, so from the perspective of the client the possible change log management does not include source code artifacts.

5.9.4 Quality Assurance

Quality Assurance is primarily on level 1. This is because quality assurance is conducted primarily after development. However, some of the tests are done automatically, which is mentioned in the description of level 3. Description of level 2 is: ”Unit testing or static code analysis is in place for some parts of the product”. This statement does not apply to the project, because there is no visibility if the vendor exercises these practices. To

Aspect in DevOps Maturity Model	**Level**	**Main reason**
Organization and Culture	1-2	Geographic distribution of teams. Separate testing team is responsible for testing.
Environments and Releases	2	Deployments are automatic. Configuration is done by hand.
Builds and Continuous Integration	1	Configuration is controlled manually. No visibility to source code version management.
Quality Assurance	1	Quality assurance is conducted primarily after development.
Visibility and Reporting	1	Code integration, unit testing, and code analysis are not visible for the client.

Table 5.2: Summary of assessing DevOps maturity using maturity model

summarize, Quality Assurance is mostly on level 1.

5.9.5 Visibility and Reporting

Visibility and Reporting are on level 1. Level 2 states: ”Code integration, unit testing and code analysis are visible for the team”. These are not visible to the client, so their existence can not be commented. Presumably, these activities are done by hand in the project. The started monitoring side project might provide better visibility for the internal functions and performance of the system like a number of orders or performance metrics.

5.10 Summary

This chapter covered DevOps adoption in a large and complex software project, in which the external vendor is doing the development work. Findings from the interviews were mapped to the categories defined in the studies of Smeds et al. (2015) and Walls (2013). If direct mapping was not possible, findings would be presented in Section 5.8. The major challenges in the project are large size and complexity, problems in the project management, occasional communication problems between the vendor and the client, poor overall quality of the software, and defects in the software development process of the vendor.

On the positive side, there have been efforts to automate the deployment process, create a monitoring process, and negotiate development and testing guidelines with the vendor. All of these efforts contribute positively to DevOps adoption. In addition, the adoption of SAFe could help managing the large and complex software project. The project is still far away from ideological DevOps model, but there has been positive progress. The ideological DevOps model might not even be suitable in a context of the project. Therefore, the company needs to remind themselves that each DevOps adoption journey is unique and analyze which practices are usable and beneficial in the project.

6 Discussion

The first challenge of this thesis was to define what DevOps is and what it encompasses. After reviewing literature available on the topic, it was clear that there was no general definition for DevOps. However, most authors emphasized that DevOps has a strong organizational and cultural aspect in addition to technical tools used to automate software delivery. It was rather difficult to decide which aspect or aspects should be emphasized in this thesis, as the limited scope of a Master's thesis means that not all aspects can be covered in depth. Therefore, the cultural and organizational aspects were chosen as the focus of this study. The reasoning behind this decision was that they are the foundation for a successful implementation of the DevOps model. Therefore, technical tools are presented cursorily.

Another challenge was that there were only a few case studies about DevOps adoption in an organization. The case study by Smeds et al. (2015) was in fact the only thorough case study carried out in an organizational context, as opposed to literature reviews. Most of Chapter 3 is based on the findings by Smeds et al. (2015). Future research would benefit from an updated literature review when more case studies have been carried out.

Eficode's (2015) and Mohamed's (2015) DevOps maturity models were reviewed in this thesis. Furthermore, Eficode's (2015) model was applied to the case company to assess the DevOps maturity of the project. Both models are suitable for obtaining a quick overview about the DevOps maturity of a project. However, they do not provide further instructions or action points for how to improve aspects that are yet not on the mature level.

Then again, the objective of these frameworks is presumably to provide an initial overview for organizations that are unfamiliar with DevOps. It should also be noted that Eficode's (2015) framework was not released as a scientific paper but as organizational marketing material. The framework was still deemed suitable for the purpose of this thesis because no other maturity models were available at the time.

Regarding the case study in this thesis, certain matters could have been executed differently. The findings are based on interviews with the personnel of the customer company in this specific system development project. This limits the viewpoint to that of the customer. Simultaneously, due the hierarchical organization of the vendor, it would have been difficult to gain access to their developers and their honest opinion on the project and its practices.

This limitation hinders the objectivity of a system development study consisting of two parties, as the researcher only hears a one-side interpretation of the state of the project. Although DevOps emphasizes organizational culture and communication, client–vendor communication was outside the scope of this study. Future research could interview both parties to gain a more holistic view. In addition, the researcher could observe meetings or agile ceremonies between the client and the vendor in order to obtain first-hand observations regarding the communication and the relationship between the parties.

The system development project studied in this case study has been ongoing for five years. At first, the waterfall development method was employed and in a sense, the adoption of DevOps model is retrofitted to the project. Retrofitting is always challenging because changing the way people or processes work is difficult. DevOps adoption is undeniably easier in a greenfield project where its methods and concepts can be applied from the very beginning. Creating working methods is always less complex than changing existing ones.

Regardless of limitations, this study lays a foundation for future studies on the DevOps adoption in an organization. This study is one of the first case studies on the client viewpoint in a DevOps adoption project where the development is done by an external

vendor. Future studies could analyze the topic from a more specific perspectives, such as how vendor selection affects DevOps adoption. Furthermore, future research could study a case project where DevOps model is implemented from day one.

7 Conclusion

The goal of this thesis was to define the aspects included in the DevOps movement. Following that, the focus moved to specifying the factors an organization should take into account when adopting DevOps. Lastly, a distinction was made between factors that contribute negatively or positively to DevOps adoption.

The first research question was: ”What is DevOps and what does the term contain?” and the second was ”Which factors should an organization consider when adopting DevOps?”. A literature review was conducted to answer the first research question. The second research question was first answered by the literature review and then strenghtened by conducting a case study, the point of which was to test if findings discovered in previous literature could be duplicated in another context.

Chapter 2 discussed and defined the core values of the DevOps movement, which are culture, automation, measurement, and sharing. Out of the four values, culture is crucially important because if the organizational culture does not fit DevOps, all automation attempts are likely to fail. Characteristics of DevOps culture include open communication, respect, trust, experimentation, constant learning, improvement, and a lack of organizational silos. These organizational traits deliver the best results when coupled with fast feedback loops. In addition to saving time, automation is used to prevent defects, create consistency, and enable self-service. Continuous integration, automated testing, and source control are examples of tools and concepts that are part of an automated deployment pipeline. Usage of technical monitoring tools support creating fast feedback loops. Careful selection of

monitoring metrics is required as choosing the wrong metrics could lead to optimization of wrong aspects and processes. Sharing success stories both within and across organizations is crucial for the successful DevOps implementation, including practical experience with tools, discoveries, and lessons.

Chapter 3 covered DevOps adoption in an organization, which may require changes in processes, personnel, and other factors. The path to successful adoption is unique to each organization. It is however possible to learn from challenges experienced by other companies DevOps adoptions and use those to plan future initiatives. In Chapter 3, factors contributing to organizational DevOps adoption were divided into capabilities, enablers, and impediments. Capabilities include continuous operations, feedback, and the ability to recover from failures. Enablers can be divided into cultural and technological ones. Examples of cultural enablers include open and effortless communication, shared responsibilities, and a supportive working environment. Technological enablers mainly focus on the need for task automation. Finally, impediments can be divided into those that affect capabilities, cultural enablers, or technological enablers. Examples of adoption impediments include hierarchical or siloed organizational structure and unclear definition of DevOps or the goals of its adoption. Chapter 3 concluded with the introduction and comparison of two DevOps maturity models.

Chapter 4 introduced the case study carried out for the thesis. The objective of the case study was to examine the status and maturity of the system development in a case organization from a DevOps viewpoint. In order to answer the research questions, qualitative research methods were employed, including interviews with selected key personnel. The case company was the Finnish subsidiary of a large multinational organization operating in the telecommunications industry. The system under study was complex, spanning across various business functions, such as order management, delivery, and network provisioning. Assessing the case study enabled the organization to examine if DevOps methods could accelerate and stabilize its system development. The case study differs from the majority of

earlier DevOps studies, as the technical development and implementation were outsourced to an external vendor.

Chapter 5 assessed the DevOps maturity of the system development project in question. The following matters were identified as the major challenges to the success of the project: the large scope and complexity of the project, problems in project management, occasional client–vendor communication problems, poor overall software quality, and defects in the software development process of the vendor. However, the study also identified changes that contributed positively to DevOps adoption, such as the automating the deployment process, creating a monitoring process, and negotiating development and testing guidelines with the vendor. In addition, the adoption of the SAFe development model has led to improvements in project management.

The research methods used in this thesis consisted of a literature review and a qualitative case study. First, DevOps and its aspects were studied by reviewing literature. After reaching an understanding about DevOps as a broad concept, DevOps adoption was studied in more detail, employing first a literature review and then applying the findings to the case study. In the case study, semi-structured open-ended interviews among key project personnel were used as a tool to study the system development project and, more specifically, its DevOps maturity.

As for the answer to RQ1, findings in Chapter 2 indicate that DevOps is a multifaceted term with no clear, widely agreed-upon definition. In addition, DevOps can be observed from various perspectives, such as technical and organizational viewpoints. In any case, DevOps appears to be a mixture of organizational culture, software development best practices, and agile ideologies. Findings in Chapter 3 underline that DevOps adoption is always a unique process and varies depending on the organization in question. However, certain matters were identified as universally problematic for the process. Therefore, organizations should carefully study their current processes and culture before establishing a clear plan for DevOps adoption.

Next, as an answer to RQ2, findings in the case study indicate that DevOps adoption is highly challenging in large and complex system development projects with a history of applying the waterfall development model. The following factors were identified as the key challenges from the DevOps adoption perspective: defects in the software development process of the vendor, lack of trust between the vendor and the client, hierarchical organizational structure, and communication problems between the management level and the personnel engaging in hands-on development work. The case study implies that retrofitting agile or DevOps methodologies to a large and complex system development project is difficult and time consuming, especially if the vendor is not skilled in agile development methods.

On the other hand, some findings indicate that an organization can expedite the software delivery process and make it more robust with rightly targeted changes. In the case study, an example of this was the automation of the deployment process, which decreased the number of errors in the deployment. As the deployment had previously been carried out manually, automation resulted in faster deployments and improved access to environments for development and testing purposes.

The case study was mainly based on the categorization of findings by Smeds et al. (2015). However, the starting points for the case study in this thesis and Smeds et al.'s (2015) study are rather different. This case study was conducted in a project where an external vendor is responsible for software development, whereas Smeds et al.'s (2015) case company itself conducted the software development. In addition, the industries were likely different, as Smeds et al.'s (2015) case company presumably offered software development services while the case company in this study outsourced software development. This observation is based on the fact that Smeds et al.'s (2015) study listed "customers may not want DevOps" as an impediment affecting DevOps adoption capabilities. Therefore, the categorization of findings is not a direct match and was not customized for a project done in collaboration with a software vendor, resulting in a limitation to the study.

Generalizing findings from a qualitative case study has its limitations. The case study was conducted in a certain type of project and environment, meaning that the results are not necessarily generalizable for other projects and environments. In addition, the findings may not be reliably transferable even to similar contexts, as projects and environments are often unique to the company in question.

Therefore, future research would benefit from a series of case studies on different organizations in order to test the generalizability of the findings. In addition, a similar case study could be conducted in a different kind of project or organizational context. Future research could also analyze topics such as vendor selection for a DevOps software project or the management of the client–vendor relationship in a DevOps project.

In conclusion, DevOps adoption is always a unique journey for any organization and its complex nature calls for careful planning. Organizations should thoroughly consider the problems they want to improve or eliminate with the adoption of a DevOps model. After the goals have been defined, suitable metrics should be selected and applied in order to measure the impact of adoption. Following the initial adoption, organizations should not pause but instead consider how to further improve their DevOps model. After all, the main aspects of DevOps include constant experimentation, improvement, and learning.

References

Apache Maven Project (2015). Maven - welcome to apache maven. https://maven.apache.org/. Online; accessed 08 October 2015.

Atlassian (2015). Continuous integration & build server - bamboo. https://www.atlassian.com/software/bamboo. Online; accessed 09 October 2015.

CFEngine (2015). Cfengine - automate large-scale, complex and mission critical it infrastructure. http://cfengine.com/. Online; accessed 08 October 2015.

Chef (2015). Chef — it automation for speed and awesomeness. https://www.chef.io/chef/. Online; accessed 08 October 2015.

Collectd (2015). collectd – the system statistics collection daemon. https://collectd.org/. Online; accessed 30 September 2015.

DevOps Dictionary (2015). CAMS - DevOps Dictionary. http://devopsdictionary.com/wiki/CAMS. Online; accessed 25 July 2016.

Eficode (2015). DevOps Quick Guides. http://devops-guide.instapage.com/. Online; accessed 16 January 2016.

Gartner (2016). Gartner IT Glossary - DevOps. http://www.gartner.com/it-glossary/devops/. Online; accessed 11 August 2016.

Git (2015). Git –fast-version-control. https://git-scm.com/. Online; accessed 08 October 2015.

Hüttermann, M. (2012). *DevOps for developers*. Apress.

Ivy (2015). Ivy - the agile dependency manager. `http://ant.apache.org/ivy/`. Online; accessed 08 October 2015.

Jenkins (2015). Welcome to jenkins ci! `http://jenkins-ci.org/`. Online; accessed 01 October 2015.

JetBrains (2015). Continuous integration for everybody - teamcity. `https://www.jetbrains.com/teamcity/`. Online; accessed 09 October 2015.

JFrog (2015). Artifactory - the open source maven repository manager. `https://www.jfrog.com/open-source/`. Online; accessed 08 October 2015.

Kim, G. (2016). The Three Ways: The Principles Underpinning DevOps. `http://itrevolution.com/the-three-ways-principles-underpinning-devops/`. Online; accessed 26 July 2015.

Liu, Y., Li, C., and Liu, W. (2014). Integrated solution for timely delivery of customer change requests: A case study of using devops approach. *International Journal of U-& E-Service, Science & Technology*, 7(2).

Lwakatare, L. E., Kuvaja, P., and Oivo, M. (2015). Dimensions of devops. In *Agile Processes, in Software Engineering, and Extreme Programming*, pages 212–217. Springer.

Mercurial (2015). Mercurial scm. `https://www.mercurial-scm.org/`. Online; accessed 08 October 2015.

Mikita, D., Dehondt, G., and Nezlek, G. (2012). The deployment pipeline. In *Proceedings of the Conference on Information Systems Applied Research ISSN*, volume 2167, page 1508.

Minick, E. (2015). Surprise! Broad Agreement on the Definition of DevOps. http://devops.com/2015/05/13/surprise-broad-agreement-on-the-definition-of-devops/. Online; accessed 25 July 2015.

Mohamed, S. I. (2015). Devops shifting software engineering strategy value based perspective. *IOSR Journal of Computer Engineering*, 17:51–57.

Munin (2015). Munin. http://munin-monitoring.org/. Online; accessed 30 September 2015.

Nagios (2015). Nagios overview. https://www.nagios.org/about/overview/. Online; accessed 30 September 2015.

Oehrlich, E. (2015). DevOps Now With CALMSS. http://blogs.forrester.com/eveline_oehrlich/15-03-02-devops_now_with_calmss. Online; accessed 25 July 2015.

Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M. B. C., and Bush, M. (1993). Key practices of the capability maturity model version 1.1.

Pollet, T., Maas, G., Marien, J., and Wambecq, A. (2006). Telecom services delivery in a soa. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 5–pp. IEEE.

Puppet Labs (2015). What is puppet? https://puppetlabs.com/puppet/what-is-puppet. Online; accessed 08 October 2015.

Riley, C. (2015). Metrics for DevOps. http://devops.com/2015/01/26/metrics-devops/. Online; accessed 08 August 2016.

Robot Framework (2015). Robot framework - generic test automation framework for

acceptance testing and atdd. http://robotframework.org/. Online; accessed 01 October 2015.

Scaled Agile guide (2016a). DevOps - Scaled Agile Framework. http://www.scaledagileframework.com/devops/. Online; accessed 22 May 2016.

Scaled Agile guide (2016b). Scaled Agile Framework - SAFe for Lean Software and System Engineering. http://www.scaledagileframework.com/. Online; accessed 10 May 2016.

Scaled Agile guide (2016c). System Team - Scaled Agile Framework. http://www.scaledagileframework.com/system-team/. Online; accessed 22 May 2016.

Selenium (2015). Selenium - web browser automation. http://www.seleniumhq.org/. Online; accessed 01 October 2015.

Smeds, J., Nybom, K., and Porres, I. (2015). Devops: A definition and perceived adoption impediments. In *Agile Processes, in Software Engineering, and Extreme Programming*, pages 166–177. Springer.

Sonatype (2015). Thenexus - a community project. http://www.sonatype.org/nexus/. Online; accessed 08 October 2015.

Subversion (2015). Apache subversion. https://subversion.apache.org/. Online; accessed 08 October 2015.

Swartout, P. (2014). *Continuous Delivery and DevOps–A Quickstart Guide*. Packt Publishing Ltd.

theagileadmin.com (2011). What is devops? http://theagileadmin.com/what-is-devops/. Online; accessed 22 September 2015.

Vagrant (2015). Vagrant - development environments made easy. https://www.vagrantup.com/. Online; accessed 08 October 2015.

Walls, M. (2013). *Building a DevOps culture*. " O'Reilly Media, Inc.".

Wettinger, J., Breitenbücher, U., and Leymann, F. (2014). Devopslang–bridging the gap between development and operations. In *Service-Oriented and Cloud Computing*, pages 108–122. Springer.

Willis, J. (2010). What devops means to me. https://www.chef.io/blog/2010/07/16/what-devops-means-to-me/. Online; accessed 21 September 2015.

Willis, J. (2016). DevOps Culture (Part 1). http://itrevolution.com/devops-culture-part-1/. Online; accessed 25 July 2015.